TEKNIIKKA JA LIIKENNE

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

WS-BPEL-LIIKETOIMINTAPROSESSIT SOA-ARKKITEHTUURISSA

Työn tekijä: Heimo Tiihonen
Työn ohjaaja: Simo Silander
Työn ohjaaja: Matti Jänne

Metropolia

ALKULAUSE

Tämä opinnäytetyö tehtiin Accenture Technology Solutions Oy:lle. Haluan kiittää ohjaajiani Simo Silanderia ja Matti Jännettä heidän neuvoistaan, joiden ansiosta lopputuloksesta tuli paljon parempi.

Helsingissä 11.5.2011

Heimo Tiihonen

Metropolia

OPINNÄYTETYÖN TIIVISTELMÄ

Työn tekijä: Heimo Tiihonen

Työn nimi: WS-BPEL-liiketoimintaprosessit SOA-arkkitehtuurissa

Päivämäärä: 11.5.2011 **Sivumäärä:** 62 s. + 5 liitettä

Koulutusohjelma: Tietotekniikka **Ammatillinen suuntautuminen:** Ohjelmistotekniikka

Työn ohjaaja: lehtori Simo Silander

Työn ohjaaja: johtava konsultti Matti Jänne

Tämän työn tarkoituksena on toimia esiselvityksenä WS-BPEL-liiketoimintaprosessien käytettävyydelle eräässä työn tilaajan asiakasprojektissa. Työssä käydään läpi WS-BPEL:in suhdetta SOA-arkkitehtuuriin, SOA:n suunnitteluperiaatteita ja vertaillaan sitä perinteisiin tapoihin toteuttaa hajautettuja järjestelmiä.

SOA:n sanotaan olevan arkkitehtuuri, joka perustuu palvelukeskeisille suunnitteluperiaatteille. SOA:n rakennuskivinä ovat mahdollisimman yleiskäyttöiset palvelut, jotka eivät pidä sisällään yksittäiseen liiketoimintaprosessiin liittyvää liiketoimintalogiikkaa. Liiketoimintaprosessikohtainen logiikka voidaan toteuttaa WS-BPEL-kuvauskieltä käyttämällä orkestrointipalvelussa, joka käyttää yleiskäyttöisiä palveluita toteuttamaan tarvitsemansa tehtävät.

WS-BPEL on IBM:n ja Microsoftin yhteistyössä kehittämä XML-pohjainen kieli suoritettavien liiketoimintaprosessien kuvaamiseen. Sen uusin versio on OASIS:n vuonna 2007 standardoima WS-BPEL 2.0. Kieltä voi käyttää sekä järjestelmien integroimiseen Web Services -rajapinnan yli että SOA:n orkestrointipalvelukerroksen toteuttavana teknologiana.

Työn tilaajaa kiinnosti erityisesti WS-BPEL-liiketoimintaprosessien siirrettävyys eri toimittajien WS-BPEL-moottoreiden välillä. Osana tätä työtä toteutettiin BluePrint5-esimerkkiprosessin siirtäminen Oraclen GlassFish ESB:ltä Apachen ODE:lle. Siirto saatiin toteutettua onnistuneesti, vaikka sitä hidastivatkin molempien WS-BPEL-moottoreiden pienet poikkeavuudet WS-BPEL 2.0 -standardista. Siirrettävyyden helpottamiseksi on tärkeää noudattaa yksityiskohtaisesti WS-BPEL-standardia eikä tukeutua toimittajien standardista poikkeaviin ratkaisuihin.

Avainsanat: SOA, WS-BPEL, liiketoimintaprosessi, orkestrointi, palvelukeskeisyys

Metropolia

ABSTRACT

Name: Heimo Tiihonen

Title: WS-BPEL Business Processes in the SOA-architecture

Date: 11.5.2011 **Number of pages:** 62 pages + 5 appendices

Department: Information Technology **Study Programme:** Software Engineering

Instructor: Senior Lecturer Simo Silander

Supervisor: Manager Matti Jänne

The purpose of this work is to be a pre-investigation for the usability of WS-BPEL business processes in one of the client's projects.

It can be said that SOA is an architecture that is based on the service oriented principles. The building blocks of the SOA-architecture are the universal services that are not part of any specific business process. The business process specific logic can be implemented using WS-BPEL language to form an orchestration service, which uses universal services to complete its tasks.

WS-BPEL is a XML-based language which is used to describe executable business processes. Its newest version is WS-BPEL 2.0, which was standardized by OASIS in 2007. WS-BPEL can be used for system integration with Web Services and to implement the orchestration layer of the SOA-architecture.

The orderer of this work was especially interested in the portability of the WS-BPEL business processes between the products of different vendors. Part of this work is to port an example process between Oracle GlassFish ESB and Apache ODE. The porting was successful even though it was hindered by small deviations from the WS-BPEL-standard. To secure the porting of WS-BPEL processes it is essentially important to follow strictly the WS-BPEL 2.0 standard and restrain from using vendor-specific features.

Keywords: SOA, WS-BPEL, business process, orchestration, service-orientation

SISÄLLYS

1 JOHDANTO

Yksi tietojärjestelmien suurimmista haasteista on aina ollut eri järjestelmien välinen vuorovaikutus. Perinteisiä järjestelmäintegraatioiden mahdollistajia ovat olleet tiedostot, joita on ennen tietoverkkojen yleistymistä kuljetettu jonkin fyysisen median kuten disketin tai magneettinauhan avulla paikasta toiseen. Tiedostojen tiedot on ajettu sisään kohdejärjestelmään yleensä eräajojen avulla. Lähi- ja kaukoverkot tekivät mahdolliseksi järjestelmille yhteiset etätietokannat sekä erilaiset etäkutsutekniikat, joiden avulla järjestelmät saattoivat kutsua toisiaan niiden tarjoamien funktiorajapintojen mukaisesti.

Etäkutsutekniikoiden heikkoutena on kuitenkin teknologiariippuvaisuus eri osapuolten välillä ja palomuuriongelmat. Etäkutsutekniikat tarvitsevat yleensä toimiakseen runsaasti tietoliikenneportteja, kun taas yritysten palomuurit useimmiten avasivat hyvin rajatun määrän portteja, usein ehkä vain HTTP-liikennettä varten portin 80. [1.]

Useimmilla yrityksillä on hyvin sekalainen sovellusinfrastruktuuri, tarkoittaen useita eri toimittajia, teknologioita ja alustoja. Siksi järjestelmien yhdistämisen tarve on saanut viimeisen vuosikymmenen aikana useat eri toimittajat kehittämään omia integrointiratkaisujaan. Näistä käytetään yleisnimitystä EAI (Enterprise Application Integration). Perinteisten EAI-tuotteiden ongelmana on se, että ne ovat yleensä tiukasti lukittuja yhteen toimittajaan ja että integrointikomponentit ovat tiukasti sidottuja tarkoittaen sitä, että kaikkien osapuolten pitää käyttää samaa integrointiteknologiaa. [2.]

Web Services -teknologia syntyi kasvavasta tarpeesta kommunikoida eri alustoille ja teknologioilla toteutettujen sovellusten välillä Internetin yli. Tämän mahdollisti integrointirajapintojen kääriminen SOAP-viesteillä kommunikoiviksi palveluiksi. Tästä sai alkunsa SOA eli palvelukeskeinen arkkitehtuuri. SOA:n keskeisenä ajatuksena on suunnitella järjestelmät liiketoimintaprosessien ympärille rakentuvina, uudelleenkäytettävinä palveluina, joiden kommunikaatio perustuu viesteihin eikä operaatioihin. SOA ei itsessään ole mikään teknologia, vaan pikemminkin ajattelumalli, mitä voidaan käyttää hyväksi tehtäessä palvelukeskeisiä sovelluksia.

SOA-ajattelun kehittyessä huomattiin pian, että tarvitaan jokin yleinen tapa yhdistellä Web Services -palveluita toisiinsa. Näin saivat alkunsa erilaiset lii-

ketoimintaprosessikielet kuten BPML, XLANG, WSFL ja sittemmin WS-BPEL [3]. WS-BPEL (Web Services Business Process Execution Language) on usean eri organisaation yhteistyönä kehittämä kieli, jolla kuvataan ja toteutetaan liiketoimintaprosesseja.

Tässä työssä pyritään vastaamaan seuraaviin kysymyksiin:

1) Mitä käsitteet SOA ja WS-BPEL pitävät sisällään?

2) Miten WS-BPEL:iä käytetään SOA-arkkitehtuurissa?

3) Mitä lisäarvoa SOA ja WS-BPEL voisivat antaa perinteisiin hajautettuihin järjestelmiin verrattuna?

4) Kuinka siirreltäviä WS-BPEL-prosessit ovat eri toimittajien tuotteiden välillä?

Työssä käsitellään lisäksi WS-BPEL:in suhdetta BPMN:n (Business Process Modeling Notation) ja ESB:n (Enterprise Service Bus) kanssa sekä esitellään joitakin yleisiä WS-BPEL-tuotteita.

Työn tilaajana toimi Accenture Technology Solutions Oy. Työtä on tarkoitus käyttää esiselvityksenä WS-BPEL:in hyödyntämisestä eräässä heidän asiakasprojekteistaan.

2 SOA

SOA (*Service Oriented Architecture)* eli palvelukeskeinen arkkitehtuuri on liiketoimintasovelluksien tekemiseen suunniteltu arkkitehtuuri, missä toisistaan riippumattomat palvelut kommunikoivat keskenään viestien avulla osana suurempaa kokonaisuutta, *liiketoimintaprosessia* [3; 4]. Palvelukeskeisyys tähtää löyhään kytkentään käyttöjärjestelmien, ohjelmointikielien ja muiden teknologioiden välillä.

SOA:n lähtökohtana on läheinen suhde liikemaailmaan. Palvelut suunnitellaan suorittamaan itsenäisiä liiketoiminnallisia tehtäviä, mitkä eivät välttämättä liity mihinkään yksittäiseen liiketoimintaprosessiin. Näitä uudelleenkäytettäviä palveluita voidaan siten tarpeen mukaan koota yhteen useissa yrityssovelluksissa. SOA pyrkii myös löyhään kytkentään liiketoimintalogiikan (Business Logic) ja työkalulogiikan (Utility Logic) välillä erottamalla ne omiksi palvelukerroksikseen. [3; 4; 5, s.334; 6, s.166.]

Liiketoimintalogiikka on johdettu yleensä liiketoiminta-analyyseistä, kuten vuokaavioista, liiketoimintaprosessien määrittelyistä ja loogisista tietomalleista. Yleisesti kaikkea, mikä esittää organisaation tapaa hoitaa liiketoimintaansa, voidaan pitää liiketoimintalogiikkana. SOA:ssa ollaan kiinnostuneita liiketoimintalogiikasta, joka on automatisoitavissa palveluiksi. Liiketoimintalogiikan automatisointiin liittyy paljon prosessointia, joka ei liity varsinaiseen liiketoimintalogiikkaan millään tavalla. Tällaista liiketoimintaan liittymätöntä logiikkaa kutsutaan työkalulogiikaksi. [6, s. 166.]

SOA:lle ei ole olemassa yhtä standardia määritelmää vaan monet eri tahot kuten standardointijärjestöt OASIS ja Open Group ovat muodostaneet omat määritelmänsä SOA:sta [7; 8]. On kuitenkin olemassa joukko yleisesti hyväksyttyjä periaatteita, jotka muodostavat palvelukeskeisyyden ajattelumallin. SOA:n voidaan sanoa olevan arkkitehtuuri, joka perustuu palvelukeskeisyyden periaatteille (luku 2.2). Vain niiden noudattamisen kautta voidaan olla varmoja siitä, että lopputulos on palvelukeskeisyyden tavoitteiden mukainen. Näihin tavoitteisiin tutustutaan seuraavaksi. [9, s.70; 5, s.40.]

2.1 Palvelukeskeisyyden tavoitteet

Palvelukeskeisyydelle on asetettu neljä päätavoitetta, joiden on havaittu tuottavan suoria positiivisia vaikutuksia tehtyjen investointien katteeseen, hallinnolliseen ketteryyteen ja IT-osaston työmäärään. [5, s. 55-65.]

Lisääntynyt luontainen yhteentoimivuus

Palvelukeskeisen suunnittelun tavoitteena on lisätä palveluiden välistä luontaista yhteentoimivuutta. Luontainen yhteentoimivuus tarkoittaa saumatonta palveluiden välistä yhteistyötä ilman erillisiä välittäjiä (broker). Jos kaikki yrityksen sisäiset palvelut ovat luontaisesti yhteentoimivia, päästään eroon järjestelmien integroinnista ja sen mukanaan tuomista hankaluuksista. Yhteentoimivuus lisääntyy lähinnä yhtenäisestä palvelukeskeisten suunnitteluperiaatteiden ja -standardien käyttämisestä.

Lisääntynyt federaatio

Federoituneessa IT-ympäristössä resurssit ja sovellukset toimivat yhteistyössä mutta säilyttävät autonomian ja itsehallinnon. Tämä tarkoittaa esimerkiksi palveluiden kohdalla sitä, että jokaisella palvelulla on täysin oma toteutuksensa ja omat resurssinsa (kts. luku 2.2, Palveluiden anatomia). SOA tähtää yrityksen sisäiseen federaatioon standardoitujen ja koostettavien palveluiden kautta. Federaatio ei tarkoita kuitenkaan täydellistä riippumattomuutta, sillä heti, kun jokin palvelu kutsuu jotain toista palvelua jonkin osatehtävän suorittamiseen, se ei ole enää täysin itsenäinen kokonaisuus vaan riippuvainen toisen palvelun toiminnasta.

Lisääntynyt toimittajariippumattomuus

SOA:n palveluiden tulee olla toimittajariippumattomia. Palveluiden tulee pystyä kommunikoimaan keskenään riippumatta siitä, millä teknologialla tai minkä alustan varaan ne ovat rakennetut. Siksi SOA pohjautuu voimakkaasti standardoituihin palvelusopimuksiin ja toteutuksen abstraktioon.

Liiketoiminnan ja tekniikan lisääntynyt linjautuminen keskenään

Sovellusten yleisenä haasteena on niiden pitäminen linjassa liiketoimintatarpeiden kanssa, kun liiketoiminnan luonne tai suunta muuttuu. SOA vastaa tähän haasteeseen oikeita liiketoimintamalleja esittävien palvelukerrosten

avulla. Näin moni olemassa oleva liiketoimintalogiikan esitysmalli voidaan suoraan toteuttaa fyysiseksi palveluksi. Tällä tavalla liiketoimintamaailma ja tekninen toteutus lähentyvät toisiaan olennaisesti ja liiketoimintamuutosten aiheuttamat muutokset ja kustannukset ovat paremmin arvioitavissa.

2.2 Palvelukeskeisyyden suunnitteluperiaatteet

SOA ei lähtökohtaisesti ota kantaa sen toteuttavaan teknologiaan, vaan on abstrakti ajattelumalli. Tällä hetkellä käytetyin teknologia SOA:n toteuttamisessa on kuitenkin Web Services. Se ei tarkoita, että kaikki ohjelmistot, jotka sisältävät Web Services -rajapintoja, noudattaisivat SOA:n periaatteita. SOA:n keskeisenä ajatuksena on palveluiden suunnittelu siten, että ne ovat mahdollisimman yleisiä ja uudelleenkäytettäviä. Se, että jollekin palvelulle tehdään Web Services -rajapinta, ei tee siitä vielä uudelleenkäytettävää tai yleistä. Monesti tällaisessa tilanteessa ei oteta huomioon sitä, että palvelun huonon suunnittelun takia voidaan joutua tekemään kompromisseja useamman palvelukeskeisyyden periaatteen kanssa. Joissakin tapauksissa se on kuitenkin perusteltua, esim. uudelleenkäytettävyyden osalta. Usein olemassa olevia palveluita ei kuitenkaan käytetä SOA:ssa sellaisinaan, vaan niistä muodostetaan pienempiä, uudelleenkäytettäviä kokonaisuuksia. [5; 9.]

SOA-ajattelumallin toteuttavien palveluiden tulisi noudattaa seuraavia yleisiä palvelukeskeisiä suunnitteluperiaatteita. [9, s.70; 5, s.40; 7.]

Standardoidut palvelusopimukset

Standardoiduilla palvelusopimuksilla tarkoitetaan joukkoa sopimuksia, jotka määrittävät palvelun toiminnallisuudet, tietorakenteen ja mahdolliset säännöt. Nämä sopimukset olisi hyvä standardoida yrityksen sisällä, jotta saavutettaisiin luontainen yhteentoimivuus palveluiden kesken. Jos SOA:n toteutuksessa käytetään Web Services -teknologiaa, palvelun toiminnallisuudet eli sen tarjoama API kuvataan WSDL-dokumentin (Web Service Definition Language) avulla. Palvelun tietorakenne määritellään XML-skeemassa ja mahdolliset säännöt WS-Policy-tiedostossa.

Palveluiden löyhä kytkentä

Palveluiden löyhällä kytkennällä tarkoitetaan sitä, että palveluiden käyttäjät ovat kytkeytyneitä palvelusopimukseen, eivät varsinaisesti palvelun toteu-

tukseen. Teknologia, jolla palvelu on toteutettu voi muuttua ilman että käyttäjän tarvitsee muuttaa omaa toteutustaan. Toisin kuin esimerkiksi toimittajariippuvaiset etäkutsutekniikat SOA-palvelut eivät ole riippuvaisia tietyn toimittajan teknologiasta vaan standardimuotoisista viesteistä, jotka välitetään eri osapuolten kesken jonkin yleiskäyttöisen protokollan avulla.

Palveluiden abstrahointi

Palveluiden abstrahoinnilla tarkoitetaan sitä että palvelun tulisi piilottaa toteutuksensa mahdollisimman hyvin. Ainoastaan ne yksityiskohdat, jotka palvelun käyttäjän tarvitsee tietää, tulisi julkaista palvelusopimuksessa. Esim. palvelun toteuttamiseen käytettyä teknologiaa tai alustaa ei pidä julkaista.

Palveluiden uudelleenkäytettävyys

Yhtenä SOA:n tärkeimmistä tavoitteista on se, että tiettyä tarkoitusta varten olisi olemassa vain yksi yleinen palvelu, joka ei tiedä mitään isäntäprosessistaan. Tällaista yleistä palvelua voidaan käyttää uudelleen useissa eri liiketoimintaprosesseissa. Kun sitten esim. jostain lakimuutoksesta johtuen useisiin eri liiketoimintaprosesseihin tulee systemaattinen muutos, tarvitsee muutos ideaalisessa tapauksessa tehdä vain yhteen palveluun. Tämä redundanssin poistaminen tuo mukanaan runsaasti etuja skaalautuvuuden ja ylläpidettävyyden muodossa [4]. Myös kokonaisia liiketoimintaprosesseja on mahdollista käyttää uudelleen aliprosesseina muissa prosesseissa.

Palveluiden autonomia

Palvelun autonomialla tarkoitetaan sitä, kuinka korkealla tasolla palvelu pystyy kontrolloimaan tarvitsemiaan resursseja. Jos palvelu käyttää esimerkiksi tietokantaa, mikä on jaettu monelle eri sovellukselle, palvelun autonomia on alhainen. Jos sillä yksin taas on pääsy käyttämiinsä resursseihin, sen autonomian taso on korkea. Palvelun alhaisesta autonomiasta seuraa alhainen luotettavuus, sillä käytettävän resurssin muut riippuvuudet eivät ole kontrolloitavissa. Lyhyesti sanottuna palvelulla tulisi olla käyttämiensä resurssien kontrolli yhtenäisyyden ja luotettavuuden maksimoimiseksi.

Palveluiden tilattomuus

Palveluiden tilattomuuden periaate tähtää tilatiedon minimoimiseen palvelussa. Palvelun ei tulisi säilyttää tilatietoa, sillä se suorittaa vain osan liike-

toimintaprosessista. Jos palvelu käyttää suuria määriä muistia, sen ja sitä pyörittävän palvelimen skaalautuvuus kärsii. Tilatiedon säilyttäminen rajoittaa myös uudelleenkäytettävyyttä. Tilatiedon säilyttämisen ja koordinoinnin tulisi tapahtua palveluita koordinoivan tahon eli liiketoimintaprosessin toimesta.

Palveluiden löydettävyys

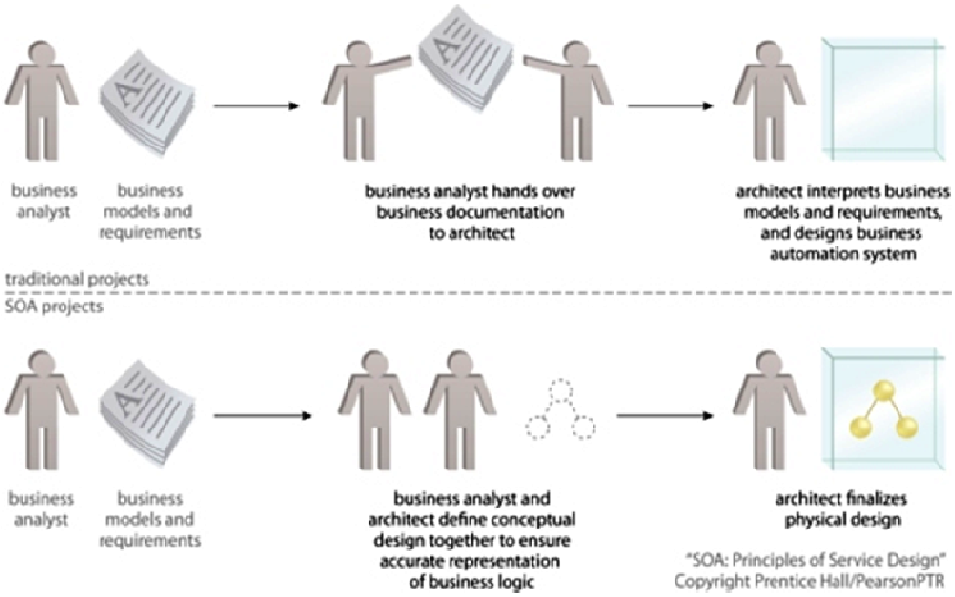
Palveluiden löydettävyyden periaate tähtää palveluiden helppoon löydettävyyteen sekä ohjelmoijien että sovellusten näkökulmasta. Siksi SOA:ssa palvelut useimmiten listataan palvelurekisteriin. Jos käytetään Web Services -teknologiaa, kysymyksessä on UDDI (Universal Description Discovery and Integration). Se on käytännössä palveluluettelo, jossa kuvataan palvelun rajapinta ja yleinen toiminnallisuus. Se sisältää myös palvelun standardoidun palvelusopimuksen. Teknisen metadatan lisäksi kuvauksen pitäisi sisältää myös ihmisluettavaa tietoa. UDDI:n avulla voidaan dynaamisesti ajonaikaisesti valita tietyn palvelusopimuksen toteuttava palvelu.

Palveluiden koostettavuus

Palveluiden koostettavuus tarkoittaa samaa kuin palveluiden yhdisteltävyys. SOA-sovellukset muodostetaan yhdistelemällä useita eri palveluita keskenään. Mitä paremmin palvelun suunnitteluvaiheessa on otettu koostettavuus huomioon, sitä useammassa sovelluksessa kyseistä palvelua voidaan tulevaisuudessa käyttää. Palveluiden koostamisessa on otettava huomioon myös koostavien palveluiden eli kontrolleripalveluiden autonomian tason laskeminen. Tämän takia palvelukerrosten lukumäärä tulisi pitää alhaisena.

2.3 Palvelukeskeinen analyysi

Jotta yritykset pystyisivät mahdollisimman tehokkaasti kehittämään standardoituja palveluita palvelurekisteriinsä, on suositeltavaa ottaa käyttöön erityinen SOA-metodologia. Siinä liiketoiminta-asiantuntijat (business analysts) ja sovellusarkkitehdit tekevät perinteistä ohjelmistoprojektia tiiviimmin yhteistyötä, jotta dokumentoitu liiketoimintamalli ja palvelun toteutus vastaisivat toisiaan mahdollisimman tarkasti (kuva 1). [10.]



Kuva 1. Liiketoiminta-asiantuntijoiden ja sovellusarkkitehtien välisen yhteistyön muuttuminen SOA-projektissa [10]

Palveluorientoitunut analyysi on iteratiivinen prosessi, jonka lopputuloksena syntyy palvelurekisterin tekninen suunnitelma, joka koostuu joukosta palvelukandidaatteja. Vaikka liiketoiminta-asiantuntijoiden ja sovellusarkkitehtien välinen yhteistyösuhde ei ole vain SOA-projektille tyypillinen, niin analyysiprosessin luonne ja laajuus on. Tarkempi analyysiprosessin kuvaus ohitetaan, mutta siitä voi lukea esimerkiksi Thomas Erlin teoksesta ”SOA Principles of Service Design”. [5; 10.]

2.4 Palvelukeskeisyyden juuret

SOA ei ole varsinaisesti mikään uusi vallankumouksellinen keksintö, vaan ennemminkin monien muiden menetelmien kuten hajautettujen järjestelmien ja modulaarisen ohjelmoinnin pohjalta syntynyt uusi kehityksen askel [5, s. 96; 11]. Ohjelmistoteollisuuden sanotaankin olleen matkalla kohti palvelukeskeisyyttä jo yli 20 vuoden ajan [4, s. 1]. Seuraavaksi käydään lyhyesti läpi joukko SOA:n kehitykseen vaikuttaneita ajattelumalleja ja teknologioita. [5, s. 88-108; 6, s. 96-100.]

Hajautetut järjestelmät

Hajautettujen järjestelmien ideana oli päästä eroon palvelin-asiakas-mallin vaikeasti ylläpidettävistä asiakassovelluksista komponenttimallilla, missä yhä suurempi osa prosessoinnista siirrettiin palvelinkomponenttien vastuulle. In-

ternetin myötä asiakassovellus korvattiin täysin internetselaimella, jolloin käytännössä kaikki sovelluslogiikka siirtyi palvelinpuolelle.

Palveluorientoitunut arkkitehtuuri näyttää ulkoisesti hyvin samanlaiselta hajautetun internet-arkkitehtuurin kanssa. Sovelluslogiikka sijaitsee palvelimella ja on toteutettu komponenttien avulla. Eroavaisuudet löytyvät palvelukeskeisen suunnittelun periaatteista, joiden avulla komponentit ryhmitellään palveluiksi tarjoamaan joukko toiminnallisuuksia. Lisäksi tapa, jolla palvelut kommunikoivat, on erilainen. Perinteisesti hajautetuissa järjestelmissä samalla palvelimella sijaitsevat komponentit kommunikoivat toistensa API:en välityksellä niin, että usein jo suunnitteluvaiheessa määritellään komponenttien riippuvuudet ja ne toteutetaan kiinteästi kooditasolla. Palvelimien välillä tapahtuva kommunikaatio tapahtuu perinteisesti jonkin RPC-protokollan tai viestitysmekanismin kuten viestijonon avulla. Viestijonot vaativat saman valmistajan tuotteen viestijonon molempiin päihin. SOA:ssa palvelut kommunikoivat standardimuotoisten viestien välityksellä löyhää kytkentää hyödyntäen. [5, s. 88-108.]

Olio-ohjelmointi (Object-Oriented Programming)

Palvelukeskeistä ohjelmointia verrataan usein olio-ohjelmointiin, sillä sen periaatteet ja mallit kuuluvat yksiin palvelukeskeisen ohjelmoinnin suurimmista vaikutteista. Palvelukeskeisyyttä ei kuitenkaan tarvitse nähdä kilpailevana ajattelumallina olio-ohjelmoinnille, sillä molemmilla on oma paikkansa. Monet palveluorientoituneet järjestelmät koostuvat sekä palvelukeskeisistä palveluista että oliokeskeisistä komponenteista. [5, s. 88-108.]

Samanlaisuuksia:

- Palveluiden uudelleenkäytettävyys on suoraa jatkumoa olio-ohjelmoinnin luokkien uudelleenkäytettävyydelle.
- Palvelusopimukset on suoraan rinnastettavissa olio-ohjelmoinnin rajapintoihin.
- Palvelun abstrahointi on läheistä sukua olio-ohjelmoinnin olioiden abstrahoinnille rajapintoja ja kapselointia käyttämällä.
- Palveluiden koostettavuus muistuttaa läheisesti olio-ohjelmoinnin aggregaattia ja kompositiota, vaikka palvelut koostetaankin löyhää kytkentää käyttäen.

Erilaisuuksia:

- Olio-ohjelmoinnissa ohjelmistoyksiköiden välille syntyy perinnän kautta paljon tiukempia kytkentöjä. Palvelukeskeisessä ohjelmoinnissa ei periytymisen käsitettä ole olemassa, vaan jokainen palvelu on täysin itsenäinen kokonaisuus.
- Autonomia on korostuneemmassa asemassa palvelukeskeisessä suunnittelussa. Aina kun palvelu käyttää jotain toista palvelua hyväkseen, laskee sen autonomian taso, sillä palvelun laatu on riippuvainen jonkin toisen palvelun laadusta. Olio-ohjelmoinnissa olioiden väliset viittaukset ja periytymisen tuomat riippuvuudet laskevat helposti autonomian tasoa.
- Olio-ohjelmoinnin luokkaa voidaan pitää mallina, jonka avulla luodaan olioita, jotka sisältävät tietoa. Oliot ovat näin luontaisesti tilallisia. Palveluiden tilattomuus eroaa näin voimakkaasti olio-ohjelmoinnista. Esimerkiksi palvelusopimuksissa ei pitäisi kuvata lainkaan ominaisuuksia vaan pelkästään operaatioita. Olio-ohjelmoinnissa puolestaan on taas vakiintunut tapa käyttää pelkästään tietoa sisältäviä olioita (Value Object tai Data Transfer Object).
- Vaikka olio-ohjelmoinnissa rajapinnat pyritään tekemään mahdollisimman johdonmukaisiksi ja itsensä selittäviksi, on löydettävyys paljon korostuneemmassa roolissa palvelukeskeisyydessä. Palvelusopimukset tehdään mahdollisimman kommunikoiviksi, jotta löydettävyys suunnittelun- ja ajonaikaisesti olisi helppoa.

Web Services

Vaikka SOA on täysin riippumaton sen toteuttavasta teknologiasta, on Web Services kuitenkin yksi sen kehitykseen eniten vaikuttaneista tekijöistä. Monet palvelukeskeisen ohjelmoinnin periaatteista, kuten palveluiden abstrahointi, löyhä kytkentä ja koostettavuus, ovat muotoutuneet Web Services -teknologian kautta. Itse asiassa Web Services -teknologia toteuttaa luontaisesti puolet palvelukeskeisistä suunnitteluperiaatteista. [5, s. 88-108.]

Ne neljä suunnitteluperiaatetta, joita Web Services ei luontaisesti toteuta, ovat seuraavat:

- palveluiden uudelleenkäytettävyys
- palveluiden autonomia

- palveluiden tilattomuus
- palveluiden löydettävyys.

Business Process Management (BPM)

BPM on tapa hallinnoida liiketoimintaa, mikä keskittyy liiketoimintaprosessien optimointiin. Sen tavoitteena on kehittää liiketoimintalogiikkaa mahdollisimman tehokkaaksi, mutta kuitenkin sopeutumiskykyiseksi ja helposti laajennettavaksi niin, että sitä voi muokata liiketoimintamuutosten tapahtuessa.

Liiketoimintaprosessien kuvaamiseen voidaan käyttää monia kuvauskieliä. Yksi niistä on Business Process Modeling Notation (BPMN), josta on kerrottu tarkemmin luvussa 3.5. Sillä kuvatut liiketoimintaprosessit ovat käännettävissä suoritettaviksi WS-BPEL -prosesseiksi. WS-BPEL -prosessit ovat XML-muotoisia prosessikuvauksia, jotka suoritetaan WS-BPEL -moottorin avulla. [5, s. 88-108.]

Enterprise Application Integration (EAI)

EAI syntyi vastauksena yritysten integrointiongelmiin 1990-luvun loppupuolella. Siihen asti sovelluksia oli integroitu toisiinsa usein P2P-tyylisesti, mikä johti ongelmiin vakauden, laajennettavuuden ja riittämättömän yhteensopivuuden muodoissa. [5, s. 88-108.]

EAI on välisovelluskerros, mikä mahdollistaa sovellusten abstrahoinnin sovittimien (adapter), välittäjien (broker) ja orkestrointimoottoreiden avulla. Vaikka EAI:n integraatioratkaisut osoittautuivatkin varsin toimiviksi, muodostuivat ne kuitenkin valtavan monimutkaisiksi ja kalliiksi. Lisäksi ne vaativat pitkäaikaista sitoutumista valitun EAI-toimittajan tuotteisiin. [5, s. 88-108.]

EAI:sta välittyivät SOA:an sen broker ja orkestrointimoottori-innovaatiot. Brokeria käytetään yhdistämään palveluita, jotka käyttävät erilaisia skeemoja tai viestirakenteita samankaltaisen tiedon esittämiseen. Orkestrointimoottoria taas käytetään suorittamaan liiketoimintaa sisältäviä palveluja, jotka koostuvat lukuisista muista palveluista. [5, s. 88-108.]

2.5 Palvelumallit

Käytännön sovellutukset ovat osoittaneet, että palvelut voidaan jaotella luokkiin niiden sisältämän sovelluslogiikan ja liiketoiminnallisten roolien perus-

teella. Näistä luokista käytetään nimitystä palvelumallit. Yksittäinen palvelu voi kuulua useampaan kuin yhteen palvelumalliin. Palvelumalleja käytetään kuvaamaan jonkin palvelun luonnetta. Sillä, mitä palvelumalleja noudattavia palveluita yhdistetään keskenään, on suuri merkitys koko järjestelmän arkkitehtuurin kannalta (vrt. luku 2.6 Palvelukerrokset). Käydään seuraavaksi lävitse joitakin yleisiä palvelumalleja. [9, s. 126.]

Liiketoimintapalvelumalli

Liiketoimintapalvelu (business service) on SOA:n kaikkein perustavanlaatuisin rakennuskivi. Se kapseloi tietyn joukon liiketoimintalogiikkaa toiminnalliseksi yksiköksi.

Liiketoimintapalvelua voidaan käyttää SOA:ssa eri tavoin. Kuten sanottu, sillä voidaan kääriä liiketoimintalogiikkaa toimivaksi palveluksi, mutta se voi myös esittää jotain yksittäistä liiketoiminnallista kokonaisuutta, kuten laskua tai hakemusta. Liiketoimintapalvelulla voi esittää kokonaisen liiketoimintaprosessin tai sitä voidaan käyttää osana suurempaa prosessia. Liiketoimintapalvelut voidaan siten jakaa kahteen eri palvelumalliin, tehtäväkeskeisiin (task-centric) ja kokonaisuuskeskeisiin (entity-centric) liiketoimintapalveluihin. [9; s. 127, s. 341.]

Tehtäväkeskeinen liiketoimintapalvelu sisältää tiettyyn liiketoimintaprosessiin liittyvää liiketoimintalogiikkaa, mikä rajoittaa niiden uudelleenkäytettävyyttä. Tämän tyyppisiä palveluita tarvitaan tyypillisesti silloin, kun liiketoimintaprosesseja ei ole keskitetty omaksi orkestrointikerroksekseen (kts. luku 2.6 Palvelukerrokset). [9, s. 341.]

Kokonaisuuskeskeinen liiketoimintapalvelu sisältää tietyn liiketoiminnallisen kokonaisuuden kuten laskun tai aikataulun. Tällaiset palvelut ovat erittäin uudelleenkäytettäviä ja liiketoimintaprosessiriippumattomia, sillä ne eivät pidä sisällään mihinkään yksittäiseen liiketoimintaprosessiin liittyvää liiketoimintalogiikkaa. Kokonaisuuskeskeisiä liiketoimintapalveluita kokoavat yhteen joko orkestrointikerroksen palvelut tai tehtäväkeskeiset liiketoimintapalvelut. [9, s.341.]

Työkalupalvelumalli

Mikä tahansa yleinen, uudelleenkäyttöä varten suunniteltu palvelu voidaan luokitella työkalupalveluksi (utility service). Kriteerinä on juuri työkalupalvelun uudelleenkäytettävyys eli hyödyntäminen useammassa eri sovelluksessa. Niillä on suurin autonomisuus kaikista palvelumalleista. [9, s. 127.]

Kontrolleripalvelumalli

Kontrolleripalvelua (controller service) käytetään tuomaan yhteen useita eri palveluita, mitkä yhdessä suorittavat jonkin suuremman liiketoiminnallisen tehtävän. Palveluiden koordinointi voi olla joko sen ainut tehtävä, tai sitten sen toissijainen tehtävä jonkin muun tehtävän ohella. Kontrolleripalvelut toteuttavat palvelukeskeisen koostettavuuden periaatteen. Myös kontrolleripalvelut voivat toimia osana suurempaa palvelukokonaisuutta. [9, s. 128.]

Hybridipalvelumalli

Palveluita, jotka sisältävät sekä työkalu- että liiketoimintalogiikkaa, kutsutaan hybridipalveluiksi. Tämä malli on yleisesti käytössä perinteisten hajautettujen järjestelmien arkkitehtuureissa. Tämä ei ole suositeltu suunnittelumalli palvelukerroksia muodostettaessa, mutta koska se on kuitenkin niin yleinen, on se luokiteltu omaksi mallikseen. [9, s. 339.]

Integraatiopalvelumalli

Palvelua, jonka ainoana tarkoituksena on mahdollistaa integraatio eri järjestelmien välillä, kutsutaan integraatiopalveluksi (integration service). Integraatiopalvelu toteutetaan yleensä kontrolleripalvelun omaisesti. [9, s. 339.]

Käärepalvelumalli

Käärepalveluita (wrapper service) käytetään useimmiten integraatiotarkoituksiin. Ne käärivät sisäänsä kokonaisia legacy-järjestelmiä tai niiden osia, paljastaakseen niiden toiminnallisuudet muille palveluille. Tyypillisin käärepalvelu on legacy-järjestelmän tarjoama palveluadapteri. Tällainen valmis palvelu yksinkertaisesti julkaisee legacy-sovelluksen API:n. Toinen käärepalvelutyyppi on proxy-palvelu, missä WSDL tuotetaan automaattisesti komponentin tarjoamasta API:sta. Komponentille generoitu proxy-palvelu vastaa

komponentin puolesta kommunikoinnista muiden palveluiden kanssa. [9, s. 339.]

Prosessipalvelumalli

Prosessipalvelut (process service) ovat orkestrointitekniikalla tehtyjä palveluita (kts. kappale 3.2 Orkestrointi ja koreografia), jotka koostavat muista palveluista kokonaisia liiketoimintaprosesseja. Näin ollen prosessipalvelu on ainoa palvelumalli, jonka ei tarvitse olla yleiskäyttöinen, vaan kuvaa tietyn liiketoimintaprosessin. Prosessipalveluista kerrotaan seuraavaksi lisää palvelukerrosten yhteydessä. [9, s. 335.]

2.6 Palvelukerrokset

Yrityssovelluksissa käytetty logiikka voidaan jakaa kahteen osaan, liiketoimintalogiikkaan ja työkalulogiikkaan (kts. luvun 2 alku). Molemmista voidaan muodostaa SOA:ssa palveluita, kunhan vain noudatetaan palvelukeskeisen ohjelmoinnin periaatteita. Nämä erityyppiset palvelut on kuitenkin hyvä eritellä omiksi palvelukerroksikseen, sillä siten saavutetaan löyhä kytkentä liiketoimintalogiikan ja työkalulogiikan välillä. [9, s. 334.]

Yhtenä SOA:n periaatteista on minimoida yksittäisen palvelun riippuvuudet. Esimerkiksi liiketoimintasääntöjä sisältävien palveluiden tulee reagoida näihin sääntöihin ajonaikaisesti. Tämä taas rajoittaa tämän tyyppisten palveluiden uudelleenkäyttöä ympäristöissä, jotka ovat kyseisten sääntöjen ulkopuolella. Toisaalta myös kontrolleripalvelut, jotka sisältävät toisten palveluiden yhdistämiseen tarvittavaa logiikkaa, saattavat muodostaa riippuvuuksia rakenteessaan. Tämän takia on hyvä ottaa käyttöön vielä yksi kontrolloinnista ja liiketoimintasäännöistä vastaava palvelukerros. Orkestrointitekniikka on kehitetty juuri tähän tarkoitukseen. Orkestrointipalvelukerros koostuu prosessipalveluista. Käydään seuraavaksi läpi kaikki kolme palvelukerrosta. [9, s. 335.]

Sovelluspalvelukerros

Sovelluspalvelukerros (application service layer) muodostaa SOA:lle perustuksen, joka ilmentää teknologiakohtaista toiminnallisuutta. Sovelluspalveluiden tarkoituksena on tarjota uudelleenkäytettäviä toiminnallisuuksia joko uusien tai legacy-sovellusympäristöjen avulla. Sovelluspalvelukerroksen palve-

lut voivat myös tarjota integraatiorajapinnan muiden sovellusten tai palveluiden kanssa. Kerroksen palvelut voivat olla sekoitus omia ja kolmannen osapuolen ostettuja tai vuokrattuja palveluita.

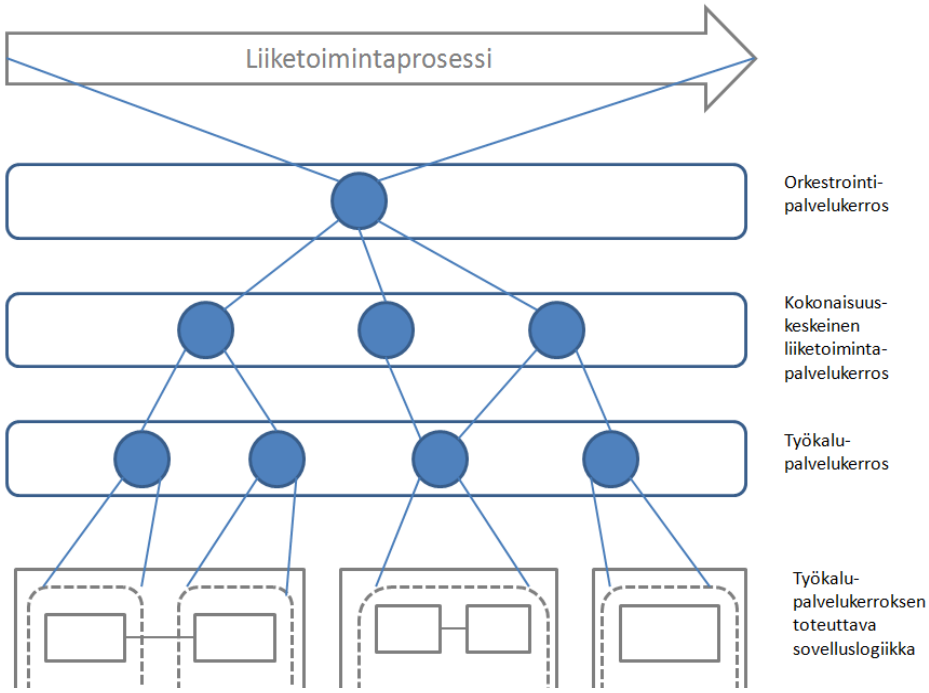
Sovelluspalvelukerros tarjoaa työkalulogiikkaa sisältäviä palveluita. Se pitää tyypillisesti sisällään työkalupalveluita, hybridipalveluita, integraatiopalveluita ja käärepalveluita. Hybridipalvelut ovat tämän palvelukerroksen erikoistapaus, sillä ne sisältävät myös liiketoimintalogiikkaa. Ne luokitellaan kuitenkin ensisijaisesti työkalupalveluiksi, koska liiketoimintapalvelukerros pitää sisällään vain puhdasta liiketoimintalogiikkaa. [9, s. 337.]

Liiketoimintapalvelukerros

Liiketoimintapalvelukerroksen (business service layer) tarkoituksena on esittää liiketoimintalogiikkaa puhtaimmassa mahdollisessa muodossa. Liiketoimintakerros koostuu pelkästään liiketoimintapalveluista. Jos työkalulogiikka on abstrahoitu sovelluspalvelukerrokseen, on todennäköistä, että liiketoimintapalvelut toimivat myös kontrolleripalveluina koostaen yhteen liiketoimintalogiikan toteuttavan sovelluspalvelukerroksen palveluita.

Orkestrointipalvelukerros

Orkestrointipalvelukerros (orchestration service layer) on korkeimman tason palvelukerros, mikä huolehtii muiden palveluiden puolesta siitä, että tehtävät suoritetaan tietyssä järjestyksessä eli sekvenssissä. Orkestrointipalvelukerros koostuu prosessipalveluista, mitkä koostavat prosessiriippumattomista yksittäisistä palveluista kokonaisia liiketoimintaprosesseja (kts. kuva 2). Prosessipalvelut ovat siis luonteeltaan myös kontrolleripalveluita. Prosessipalvelu itsessään voidaan koostaa osaksi suurempaa liiketoimintaprosessia, jos se on luonteeltaan uudelleenkäytettävä. [9, s. 344.]



Kuva 2. Prosessipalvelu, joka koostaa kokonaisuuskeskeisiä liiketoimintapalveluita, jotka koostavat työkalupalveluita. [9, s. 353]

Palvelukerrosvariaatiot

Kuvassa 2 on esitetty ideaalinen SOA-malli, missä on selvästi eroteltu liiketoiminta- ja työkalulogiikka toisistaan ja samalla maksimoitu uudelleenkäytettävyys. Prosessipalvelu sisältää kaiken prosessiin liittyvän logiikan, joka käyttää sen koostamiseen kokonaisuuskeskeisiä liiketoimintapalveluita, jotka taas koostavat työkalupalveluita suorittamaan tarvitut tehtävät. Tämä on kuitenkin harvinainen tilanne todellisessa SOA-maailmassa, sillä usein on käytössä muitakin kuin ideaalisessa mallissa käytettyjä palvelumalleja. Tästä johtuen saattaa syntyä taulukon 1 mukaisia palvelukerrosvariaatioita.

Taulukko 1. Palvelukerrosvariaatiot

Variaatio	Prosessi-palveluita	Kokonaisuus-keskeisiä liiketoi-mintapalveluita	Tehtäväkeskeisiä liiketoiminta-palveluita	Hybridi-palveluita	Työkalu-palveluita
1				X	
2				X	X
3			X		X
4		X	X		X
5	X			X	X
6	X		X		X
7	X	X	X		X
8	X	X			X

Taulukon 1 palvelukerrosvariaatioiden kuvaukset:

1. Yleinen, kun lisätään Web Services -teknologia johonkin olemassa olevaan hajautettuun järjestelmään ilman, että otetaan kantaa uudelleenkäytettävyyteen tai palvelukeskeiseen suunnitteluun.

2. Edellistä tapausta on kehitetty luomalla uudelleenkäytettäviä työkalupalveluita, joita osa hybridipalveluista käyttää hyväkseen.

3. Liiketoimintalogiikka on eristetty omaksi kerroksekseen tehtäväkeskeisillä liiketoimintapalveluilla. Ne nojautuvat täysin työkalupalveluihin sovelluslogiikan osalta.

4. Tehtäväkeskeiset liiketoimintapalvelut koostavat sekä kokonaisuuskeskeisiä liiketoimintapalveluita että työkalupalveluita.

5. Prosessipalvelu koostaa hybridi- ja työkalupalveluita liiketoimintaprosessiksi. Yleinen tilanne, kun lisätään orkestrointikerros olemassa olevan hajautetun järjestelmäarkkitehtuurin päälle.

6. Liiketoimintaprosessin logiikkaa sisältävien tehtäväkeskeisten liiketoimintapalvelujen päälle voidaan myös lisätä prosessipalvelu koostamaan sekä tehtäväkohtaisia liiketoimintapalveluita että työkalupalveluita. Näin saadaan koko liiketoimintalogiikka abstrahoitua omiksi kerroksikseen.

7. Edelliseen verrattuna on otettu käyttöön vielä kokonaisuuskeskeiset liiketoimintapalvelut, jotka tuovat mukanaan jo neljännen palvelukerroksen, mikä asettuu tehtäväkohtaisten liiketoimintapalveluiden ja

työkalupalveluiden väliin. Tehtäväkohtaiset liiketoimintapalvelut saattavat esittää aliprosesseja, mitä yhdessä kokonaisuuskeskeisten liiketoimintapalveluiden kanssa koostetaan kokonaiseksi liiketoimintaprosessiksi prosessipalvelun avulla.

8. Ideaalitapaus, jossa prosessipalvelu sisältää kaiken prosessiin liittyvän logiikan, joka käyttää sen koostamiseen kokonaisuuskeskeisiä liiketoimintapalveluita, jotka taas koostavat työkalupalveluita suorittamaan tarvitut tehtävät.

2.7 Perinteisten ja SOA-liiketoimintasovellusten vertailua

Perinteisten liiketoimintasovellusten ohjelmistokehitys on perustunut automatisoitavien liiketoimintatehtävien tunnistamiseen, vaatimusmäärittelyyn ja sitä vastaavan sovelluslogiikan rakentamiseen. Tämä on ollut hyväksi todettu tyyli tehdä liiketoimintasovelluksia ja sillä on saatu suhteellisen ennustettavissa oleva vastine investoidulle pääomalle. Ne ovat kuitenkin olleet sidottuja tiettyihin liiketoimintavaatimuksiin ja prosesseihin. Kun uusia vaatimuksia ja prosesseja syntyy, täytyy joko tehdä olemassa oleviin sovelluksiin huomattavia muutoksia tai sitten rakentaa kokonaan uusi sovellus. Vaikka tämä kertakäyttöisten sovellusten tekeminen ei olekaan paras mahdollinen tapa, on se osoittautunut oikeutetuksi tavaksi automatisoida liiketoimintaa. [5, s. 76-77.]

Perinteistä ohjelmistokehitystä tukevia syitä ovat:

- tehokkuus, sillä ohjelmiston pitää ottaa huomioon vain rajatun liiketoimintaprosessijoukon vaatimukset.
- liiketoiminta-analyysin suoraviivaisuus, sillä analyysi keskittyy vain yhteen prosessiin kerrallaan.
- suunnittelun yksinkertaisuus, sillä tarkoituksena on automatisoida vain rajattu joukko liiketoimintaprosesseja.
- nykyaikaisuus, sillä uusia sovelluksia tehtäessä voidaan ottaa aina käyttöön uusinta teknologiaa.

Perinteisen ohjelmistokehityksen haittapuolia:

- Uuden sovelluksen luominen lisää huomattavan määrän päällekkäistä toiminnallisuutta muiden sovelluksien kanssa. Täten myös käytetty pääoma ja vaiva ovat päällekkäisiä.
- Jokainen uusi sovellus lisää ylläpito- ja hallintovaatimuksia ja saattaa paisuttaa IT-osaston resursseja liiaksi niin, että IT:stä tulee huomattava kuormittaja organisaatiolle.
- Lukuisten eri sukupolvea olevien, eri teknologioilla toteutettujen sovellusten hallinnointi edellyttää usein myös yksilöllisiä arkkitehtuurillisia vaatimuksia.
- Tiettyä liiketoimintatarkoitusta varten tehtyä sovellusta ei ole useinkaan suunniteltu ottamaan huomioon yhteensopivuutta muiden sovellusten kanssa. Lukuisten tämän tyyppisten sovellusten tarve kommunikoida toistensa kanssa synnyttää monimutkaisia integraatiorakenteita joko p2p-yhteyksillä tai ottamalla käyttöön välikerrossovelluksia.

Palvelukeskeisyyden tuomat ratkaisut:

Aikaisemmin käsiteltyjen kahdeksan palvelukeskeisen suunnitteluperiaatteen johdonmukainen käyttö tuottaa mm. seuraavia ominaisuuksia:

- Sovelluksista ja liiketoimintaprosesseista riippumattoman sovelluslogiikan määrä lisääntyy.
- Sovelluskohtainen sovelluslogiikka vähenee.
- Koska samoja palveluja käytetään uudelleen useissa eri liiketoimintaprosesseissa, myös tarvittavan sovelluslogiikan kokonaismäärää vähenee.
- Kun samoja suunnittelumalleja käytetään johdonmukaisesti, syntyy kauttaaltaan yhtenäistä sovelluslogiikkaa. Kun lisäksi pidetään huolta palvelusopimusten ja niiden tietomallien standardoinnista, saavutetaan palveluille automaattinen yhteentoimivuus. Näin ollen, jos yrityksen kaikki järjestelmät on toteutettu palveluorientoituneesti, integraatioiden tarve poistuu. [5, s. 81-84.]

2.8 Tiedon sijainti ja saatavuus SOA-arkkitehtuurissa

Tähän mennessä ei ole käsitelty sitä, missä tieto sijaitsee SOA-arkkitehtuurissa ja miten sitä käsitellään. Autonomisuuden suunnitteluperiaatteen mukaisesti SOA:ssa tulisi pyrkiä palvelukohtaisiin tietovarastoihin. Kokonaisuuskeskeiset liiketoimintapalvelut kuvaavat todellisia liiketoiminnallisia kokonaisuuksia, kuten asiakas tai tilaus. Näihin kokonaisuuksiin liittyvät tiedot ovat saatavissa vastaavien palveluiden kautta. Jos tavoitellaan korkeaa autonomisuutta, voidaan jokaista palvelua varten luoda oma tietokantansa. Usein kuitenkin voidaan tyytyä siihen, että kokonaisuuskeskeiset liiketoimintapalvelut abstrahoivat sisäänsä käyttämänsä tietolähteet ja tietorakenteet. Tällöin palvelut pitävät tavallisesti sisällään välimuistia, mikä nopeuttaa huomattavasti tiedonhakuoperaatioita. Tällaisten lukuisia tietolähteitä käsittelevien palveluiden toteuttaminen käsin on kuitenkin niin hankalaa, että monia kaupallisia tiedon virtualisointiin kehitettyjä sovelluksia on tullut markkinoille. Tällaisia tuotteita kutsutaan tietokehikoiksi (Information Fabric). [12].

Tietokehikko kokoaa yhteen tietoa useista eri tietolähteistä muodostaen tiedosta yhtenäisen näkymän, joka voidaan julkaista kokonaisuuskeskeisenä liiketoimintapalveluna. Nämä palvelut muodostavat tiedonsaantikerroksen (Data Access Layer) IAAS-periaatteen (Information As A Service) mukaisesti. Tietokehikon toiminnallisuuksiin kuuluu myös kyselyiden optimointia, välimuistin ja transaktioiden hallintaa, tietoturvaa ja palvelutason monitorointia. Eräs tietokehikko on IBM InfoSphere Federation Server. [12.]

2.9 Haasteet

Vaikka SOA ratkaisee monia perinteisiä teknisiä ongelmia, tuo se myös mukanaan joukon uusia. On tärkeää tiedostaa SOA:n haasteet, sillä ne kaikki ovat ratkaistavissa hyvällä arkkitehtuurisuunnittelulla, moderneilla ajoympäristöillä ja säännönmukaisella palveluorientoituneiden suunnitteluperiaatteiden noudattamisella. [5, s. 85.]

Lisääntyneet tehokkuusvaatimukset

Yleiskäyttöisten palveluiden suunnittelu tarkoittaa yleensä sitä, että suunnitellaan palveluita, jotka osaavat tehdä yhden asian erittäin hyvin. Tämä tarkoittaa toisaalta palveluiden lukumäärän lisääntymistä, mikä puolestaan li-

sää lähetettyjen ja vastaanotettujen viestien lukumäärää, mikä taas tulee ilmi lisääntyneenä kuormana viestien käsittelystä johtuen.

Palveluiden ruuhkautuminen ja huolto

Palvelut saattavat joskus ylikuormittua ruuhkahuippujen aikana. Palvelut tulisi mitoittaa siten, että ne pystyvät selviytymään uudelleenkäytettävyyden tuomista lisähaasteista. Palveluita tulee myös huoltaa, mikä uudelleenkäytettävyyden takia vaikuttaa yleensä monen eri sovelluksen toimintaan.

Single point of failure -ongelmat

Kun yksi ja sama palvelu on osa montaa eri sovellusta, sen vikaantuminen johtaa usean eri sovelluksen kaatumiseen. Tästä johtuen on joskus syytä lisätä redundanssia tekemällä samasta palvelusta monta eri asennusta. Näin yhden asennuksen vikaantuessa voidaan ottaa jokin vara-asennuksista käyttöön.

Lisääntyneet vaatimukset palveluja isännöiville ympäristöille

Palveluiden autonomia -suunnitteluperiaate tuo lisävaatimuksia palveluita isännöiville ympäristöille, koska jokaiselle palvelulle tulee löytyä omat hallittavissa olevat resurssinsa.

Palvelusopimusten versiointiongelmat

Jos olemassa olevaa palvelusopimusta joudutaan joskus muokkaamaan, joudutaan palvelusopimusten versiointiin. Versioinnin yhteydessä joudutaan ottamaan kantaa esim. siihen, ovatko vanhaa palvelusopimusta käyttävät palvelut tuettuja uuden palvelusopimuksen puolesta ja joudutaanko palvelusopimuksia säilyttämään samanaikaisesti ja kuinka pitkään. [13.]

3 WS-BPEL-MALLINNUSKIELI

WS-BPEL (Business Process Excecution Language*)* on XML-pohjainen liiketoimintaprosessien mallinnuskieli, mikä perustuu Web Services -teknologiaan. WS-BPEL voidaan lukea jopa yhdeksi Web Services -teknologian useista laajennuksista [9]. Kielen avulla ohjelmoija kuvaa liiketoimintaprosessin, joka käyttää liiketoiminnallisten tehtävien suorittamiseen Web Services -palveluita (tästä eteenpäin Web Services -palvelu on WS-palvelu). Myös itse WS-BPEL-liiketoimintaprosessi voidaan julkaista WS-palveluna. WS-BPEL toimii ns. liimana WS-palveluiden välillä sitoen ne yhtenäiseksi järjestelmäksi ja johtaen niiden keskinäistä yhteydenpitoa [3].

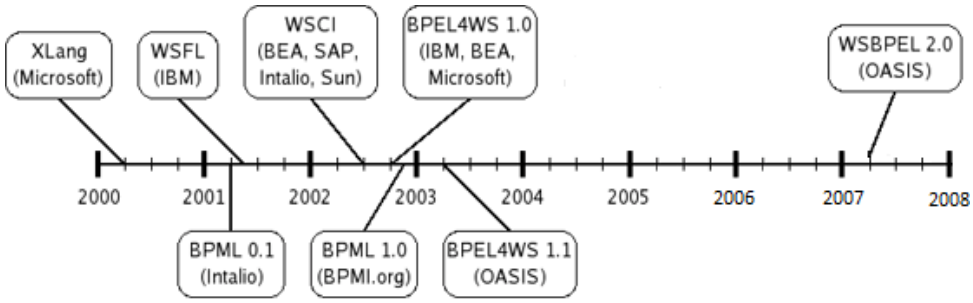
WS-BPEL:iä käytetään yrityksen sisäisesti mm. yrityssovellusten integroinnin standardisoinnissa ja integraation laajentamisessa aikaisemmin erillisiin järjestelmiin. WS-BPEL mahdollistaa myös yritysten välisen integraation liittämällä eri yhteistyökumppaneiden WS-palvelut yhtenäiseksi järjestelmäksi. [14.]

WS-BPEL ei kuitenkaan ole pelkästään integrointityökalu. Jos SOA:n palvelut ovat toteutukseltaan WS-palveluita, voi WS-BPEL toimia orkestrointipalvelukerroksen toteuttavana teknologiana, jonka vastuulla on liiketoimintaprosessien koostaminen yleiskäyttöisistä liiketoimintapalveluista (kts. luku 2.6.). Tällöin kaikki liiketoimintaprosessit on abstrahoitu omaksi palvelukerrokseksi WS-BPEL:in avulla, jolloin liiketoimintaprosessien ylläpito on helppoa. [6, s. 185-186.]

3.1 Historia

WS-BPEL sai syntynsä kun BEA Systems, IBM, and Microsoft päättivät kehittää yhteisen standardin WS-palveluiden koordinointiin omien versioidensa pohjalta. Microsoftilla oli XLANG ja IBM:llä WSFL (Web Flow Language). Vuonna 2002 julkaistiin versio 1.0 nimellä BPEL4WS. Seuraavan vuoden toukokuussa kielen kehittäneet yhtiöt luovuttivat version 1.1 OASIS:lle standardoitavaksi. [15; 9, s.568.] Vuonna 2004 OASIS ilmoitti uuden standardin nimeksi WS-BPEL 2.0. Varsinainen julkaisu tehtiin kuitenkin vasta 11. huhtikuuta 2007 [16].

WS-BPEL:in historiaan liittyy läheisesti yksi sen suurimmista kilpailijoista, BPML (Business Process Modeling Language). BPML:n kehittäminen alkoi kesällä 2000, kun 16 yritystä perustivat BPMI:n (Business Process Management Initiative). Tästä alkoi avoin kehitysprosessi, ja 18 kuukautta myöhemmin mukana oli jo 200 yritystä [17]. BPML 1.0 julkistettiin lokakuussa 2002, vain kuukausi BPEL 1.0:n jälkeen [15; 17]. Silloin BPEL4WS-nimellä tehty julkistus oli ovela veto, sillä vaikka teknologiaa ei ollut vielä standardoitu, se keräsi nopeasti kannattajia ympärilleen. Kuvassa 3 on esitetty edellä mainittujen mallinnuskielten sijoittuminen aikajanalle.



Kuva 3. Suoritettavien liiketoimintaprosessien kuvauskielten aikajana [15; 16]

BPML muistuttaa paljon BPEL:iä,mutta tuo siihen lisäominaisuuksia kuten oikeat transaktiot ja sisäkkäiset prosessit. Sitä pidetäänkin BPEL:in ylijoukkona [18].

BPML on myös suunniteltu kokonaiseksi (eng. complete) prosessikieleksi, toisin kuin BPEL. Kokonainen tarkoittaa sitä, että BPML:n avulla pystyy mallintamaan prosessin kuin prosessin, kun taas BPEL:istä puuttuu joitakin ominaisuuksia, kuten tiedon muokkaus ja laskutoimitukset, mitä varten IBM ja BEA kehittivät siihen Java-laajennuksen BPELJ [18]. Tämä on kuitenkin monien mielestä väärä suuntaus, sillä se yhdistää ”programming in the large” ja ”programming in the small" -ohjelmointikielet ja mm. rajoittaa BPEL-prosessien siirrettävyyttä. Oikeampi tapa olisi toteuttaa puuttuvat toiminnallisuudet omina WS-palveluinaan. [19.]

BPML:n ja BPEL:in välisestä taistelusta voittajana selvisi kuitenkin BPEL. Tämä on hyvä esimerkki siitä, kuinka suurten yritysten linjaukset liikuttavat maailmaa. IBM oli mukana BPML:n kehityksessä jo varhain, mutta valitsi jostain syystä lähteä Microsoftin kanssa omalle tielleen. Vuonna 2005 OMG ja BPMI yhdistyivät, ja samalla BPMI.org ilmoitti suosittelevansa BPEL:iä käy-

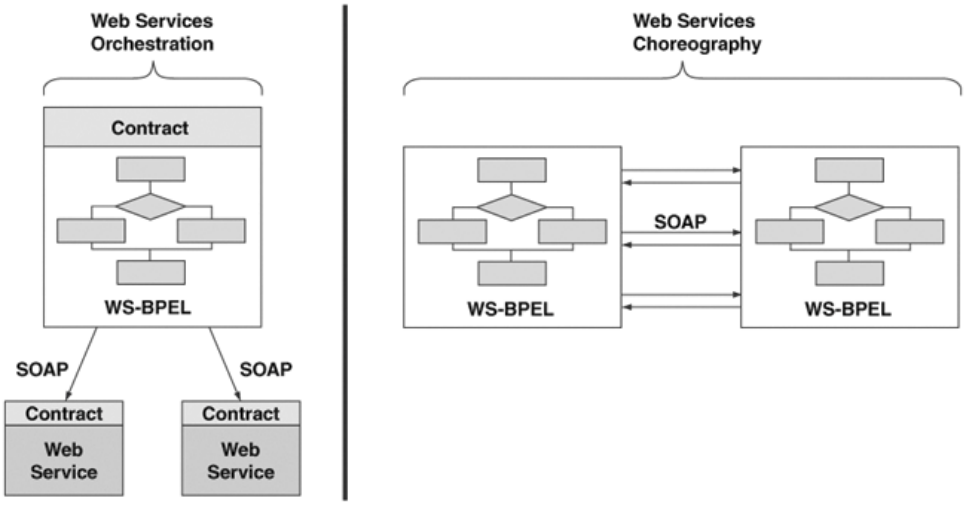
tettäviksi prossessikieleksi. BPML-standardi poistettiin käytöstä, ja sen kehitystä jatketaan tällä hetkellä nimellä BPDM (Business Process Definition Metamodel) [20]. Näyttää vahvasti siltä, että WS-BPEL on saavuttanut standardin aseman suoritettavien liiketoimintaprosessien mallinnuskielenä.

3.2 Orkestrointi ja koreografia

Orkestrointi ja koreografia edustavat kahta erilaista tapaa yhdistellä palveluita keskenään toimiviksi kokonaisuuksiksi.

Orkestrointi on yleensä yksityisissä eli yritysten sisäisissä liiketoimintaprosesseissa käytetty malli, missä jokin keskusprosessi kontrolloi palveluita ja niiden välistä kommunikointia (kuva 4). Prosessissa mukana olevien palveluiden ei tässä mallissa tarvitse tietää mitään isäntäprosessista. [14.]

Koreografia puolestaan ei tukeudu yhteen keskusprosessiin, vaan jokainen prosessissa mukana oleva palvelu tietää täsmälleen milloin, millä tavalla ja kenen kanssa keskustella (kuva 4) [14]. Tämä malli sopii erityisesti yritysten välisiin liiketoimintaprosesseihin, koska siinä mikään yritys ei yksin omista eikä kontrolloi koko liiketoimintaprosessia. [9, s.209.]



Kuva 4. Web Services: Orkestrointi ja koreografia [21]

WS-BPEL:in tapa hoitaa palveluiden koordinointia on orkestrointi. Koreografiaa varten on olemassa WS-laajennus WS-CDL (Web Services Choreography Description Language).

3.3 Abstraktit ja suoritettavat prosessit

WS-BPEL tukee kahta vaihtoehtoista tapaa liiketoimintaprosessien kuvaamiseen, abstrakteja ja suoritettavia prosesseja. Suoritettava prosessi mahdollistaa tarkkojen yksityiskohtien määrittämisen. Se seuraa orkestrointiajattelumallia ja sen voi suorittaa jollain orkestrointimoottorilla.

Abstrakti prosessi on osittain määritelty prosessi, jota ei ole tarkoitettu suoritettavaksi ja joka pitää eksplisiittisesti merkitä abstraktiksi. Abstrakti prosessi voi sisältää kaikki suoritettavan prosessin rakenteet; ne ovat siis yhtä ilmaisuvoimaisia. Suoritettavan prosessin ominaisuuksien lisäksi abstraktissa prosessissa voidaan haluttaessa piilottamaa toiminnallisia yksityiskohtia. [16.]

Abstraktilla prosessilla on kuvaava rooli, ja sillä on useampia käyttötapauksia. Yksi voi olla kaikkien suoritettavan prosessin palveluiden kuvaaminen. Toiseksi sitä voidaan käyttää prosessipohjana, joka sisältää yrityksen määrittelemät parhaat toimintamallit. Abstrakti prosessi perustuu ns. yleiseen pohjaan (common base), mitä voi tarkentaa jollain käyttötapausprofiililla. [16.]

Abstrakti prosessi määritellään abstraktiksi process-elementissä uri:lla http://docs.oasis-open.org/wsbpel/2.0/process/abstract. Pakollinen käyttötapausprofiili määritellään process-elementin abstractProcessProfile-attribuutissa profiilikohtaisen uri:n avulla. [16.]

3.4 Tekninen kuvaus

Suoritettava WS-BPEL-prosessi on ulkoisesti tavallinen WS-palvelu. Sille tulee muiden WS-palveluiden tavoin määritellä täsmällinen WSDL-kielinen rajapintakuvaus, missä määritellään mm. kaikki operaatiot sekä niiden lähettämät ja vastaanottamat viestit. WS-BPEL:issä liiketoimintaprosessiin osallistuvia WS-palveluita kutsutaan osakkaiksi (partner) [15].

Vaikka WS-BPEL-prosessit tehdäänkin yleensä jonkin graafisen mallinnustyökalun avulla, kieli on standardoitu ainoastaan XML-kuvauksen tasolla. Se ei toisin sanoen ota mitenkään kantaa graafiseen mallinnustyökalun ominaisuuksiin tai tapaan kuvata prosessia. Siksi kielen ominaisuudet on paras opiskella XML-kuvauksena, sillä se pysyy samana työkalusta toiseen.

Seuraavaksi tutustutaan tarkemmin WS-BPEL 2:0:n rakenteeseen ja esitellään sen tärkeimpiä ominaisuuksia. BPEL4WS:n versiot 1.0 ja 1.1 ohitetaan, koska WS-BPEL 2.0 on voimassa oleva, standardoitu versio. WS-BPEL 2.0 tukee WS-I Basic Profile:n versiota 1.1, ja on yhteensopiva standardien WSDL 1.1, XML Schema 1.0, XPath 1.0, XSLT 1.0 ja Infoset kanssa [16]. Yksityiskohtaisempi WS-BPEL 2.0 spesifikaatio löytyy OASIS:n nettisivuilta [16].

Koska WS-BPEL pohjautuu Web Services -arkkitehtuuriin ja on jopa luettavissa yhdeksi sen laajennuksista, tarvitaan kielen opiskelussa hyvää tuntemusta myös Web Services -arkkitehtuurista. Tässä työssä ei kuitenkaan käsitellä Web Services arkkitehtuuria, vaan se on esitietovaatimuksena. Hyvää materiaalia löytyy esim. W3.org:in sivuilta (www.w3.org/TR/ws-arch/).

3.4.1 Rajapintakuvaukset

Koska WS-BPEL-prosessi on käytännössä WS-palvelu, tulee sille määritellä rajapinta omassa WSDL-dokumentissaan. Liiketoimintaprosessin rajapintakuvaus on samanlainen kuin tavallisella WS-palvelulla seuraavia poikkeuksia lukuun ottamatta [16]:

1. Rajapintakuvaukseen tulee lisätä partnerLinkType-elementtejä kuvaamaan liiketoimintaprosessin ja osakaspalveluiden välistä kommunikaatiota. Tämä koskee myös osakaspalveluiden rajapintakuvauksia. Elementti ei kuulu WSDL 1.1 -standardiin, joten se lisätään laajennuksena (http://docs.oasis-open.org/wsbpel/2.0/plnktype). Jokainen partnerLinkType-elementti sisältää 1-2 roolin kuvaukset, jotka sisältävät roolin nimen ja siihen liitettävän porttityypin. Roolien lukumäärän määrittelee kommunikoivien palveluiden välisten kutsujen suunta: jos vain toinen osapuoli kutsuu toisen operaatioita, riittää yksi rooli. Jos molemmat osapuolet kutsuvat toistensa operaatioita, rooleja tarvitaan kaksi. Näitä roolimäärityksiä käytetään hyväksi itse liiketoimintaprosessissa. [16.]

2. Liiketoimintaprosessin rajapintakuvaukseen ei tarvitse määritellä service- ja binding-elementtejä. Liiketoimintaprosessissa palvelut voidaan määritellä dynaamisesti pelkästään porttityyppien avulla, mikä mahdollistaa liiketoimintaprosessin uudelleenkäyttämisen muiden palvelutoteutusten kanssa. [16.]

Jälkimmäinen poikkeus riippuu kuitenkin WS-BPEL -moottorin toimittajasta. Esimerkiksi Apache ODE:n WAR-distribuution kanssa tulee käyttää binding- ja services-kuvauksia, kun taas JBI-distribuutio ei kyseisiä kuvauksia tarvitse.

Liittessä 1 on esimerkki rajapintakuvauksesta, joka on sovitettu Apache ODE:n WAR-distribuutiolle.

3.4.2 Rakenne

WS-BPEL -liiketoimintaprosessi alkaa process-juurielementillä. Se sisältää aina attribuutit prosessin nimelle ja prosessissa esiintyville nimiavaruuksille. WS-BPEL -prosessin runko on yleisesti koodiesimerkin 1 kaltainen [9, s. 567].

```
<process>
        <partnerLinks>
          ...
        </partnerLinks>
        <variables>
          ...
        </variables>
        <faultHandlers>
          ...
        </faultHandlers>
        <sequence>
                <receive>
                <invoke>
                <reply>
                ...
        </sequence>
</process>
```

Koodiesimerkki 1. WS-BPEL-prosessin runko

Liitteessä 2 on esimerkki suoritettavasta WS-BPEL-prosessista. Se on osa BluePrint5-esimerkkiä, jota käydään tarkemmin läpi luvussa 5. BluePrint5 on yksinkertainen varausjärjestelmä, jolla voi varata itselleen auton, hotellin ja/tai lentolipun. Kyseisestä prosessista ovat myös muut tässä luvussa käytetyt koodiesimerkit.

Liitteessä 3 on kuvattu kaikki WS-BPEL -prosessin ylätason elementit aktiviteetteja ja näkyvyysalueita lukuun ottamatta, jotka käsitellään omissa luvuissaan (3.4.5. - 3.4.6).

3.4.3 Osakkaat

Liiketoimintaprosessin suoritukseen osallistuvat WS-palvelut eli osakkaat määritellään partnerLinks-elementin partnerLink-lapsielementtien avulla.

Osakas voi olla joko prosessin asiakas, eli se voi aktivoida prosessin, tai osakas voidaan suorittaa prosessin toimesta.

Koodiesimerkissä 2 on esitetty ReservationSystem-prosessin (liite 2) osakkaat. PartnerLink-elementillä on name-, partnerLinkType-, myRole- ja partnerRole-attribuutit. Attribuutilla myRole ilmaistaan prosessin rooli, kun prosessia kutsutaan osakaspalvelun toimesta. ReservationSystem-prosessi toteuttaa itse osakkaat reserevationPLink, airlineStatus, carReservationStatusPLink ja hotelReservationStatusPLink.

Sama partnerLink-elementti voi sisältää sekä myRole- että partnerRole-attribuutit, kun odotetaan prosessin toimivan sekä palvelun tarjoajana että kuluttajana. Tilanne on tämä esimerkiksi asynkronisen kommunikaation tapauksessa, jolloin jokin osakaspalvelu herättää prosessin, mutta ei jää odottamaan suorituksen loppuun vaan jättää lopputuloksen palauttamisen liiketoimintaprosessin vastuulle [9, s. 569].

```
<partnerLinks>
        <partnerLink name="reservationPLink"
                     partnerLinkType="ns0:reservationPLinkType"
                     myRole="reservationServicing" />
        <partnerLink name="airlineReservation"
                     partnerLinkType="ns1:AirlineReservationPLinkType"
                     partnerRole="reservingRole" />
        <partnerLink name="airlineStatus"
                     partnerLinkType="ns1:AirLineReservStatusPLinkType"
                     myRole="statusReceivingRole" />
        <partnerLink name="carReservationPLink"
                     partnerLinkType="ns2:CarReservationPLinkType"
                     partnerRole="reservingRole" />
        <partnerLink name="carReservationStatusPLink"
                     partnerLinkType="ns2:CarReservStatusPLinkType"
                     myRole="statusReceivingRole" />
        <partnerLink name="hoteReservationPLink"
                     partnerLinkType="ns3:HotelReservationPLinkType"
                     partnerRole="reservingRole" />
        <partnerLink name="hotelReservationStatusPLink"
                     partnerLinkType="ns3:HotelReservStatusPLinkType"
                     myRole="statusReceivingRole" />
</partnerLinks>
```

Koodiesimerkki 2. Osakaspalveluiden ja roolien määrittäminen prosessissa ReservationSystem

3.4.4 Muuttujat

Liiketoimintaprosessin tilan ylläpitäminen edellyttää muuttujien käyttämistä. Ne määritellään WS-BPEL:issä variables-elementin variable-lapsielementeillä. Muuttujat voivat olla tyypiltään XML-tietotyyppejä, tai WSDL-viestejä. Muuttujat määritellään aina jossain näkyvyysalueessa, ja ne kuuluvat kyseiselle näkyvyysalueelle. Jokainen muuttuja on näkyvissä vain oman näkyvyysalueensa sisällä mukaan lukien näkyvyysalueelle alistetut

alemman tason näkyvyysalueet. Process-elementin näkyvyysalueella määritellyt muuttujat ovat globaaleja muuttujia. Ylemmällä tasolla määritellyn muuttujan voi piilottaa määrittelemällä samanniminen muuttuja alemman tason näkyvyysalueelle. Muuttujia pääsee käsittelemään käytössä olevalla kysely- ja lausekekielellä, mikä on oletusarvoisesti XPath 1.0. Kielen syntaksia ei käsitellä tarkemmin tässä opinnäytetyössä.

Koodiesimerkissä 3 on esitetty ReservationSystem-prosessin käyttämät muuttujat. Muuttujat ovat tyypiltään WSDL-viestejä, jotka ovat määritelty ReservationSystem-prosessin rajapintakuvauksessa (liite 1).

```
<variables>
        <variable name="reserve"
                  messageType="ns0:reserveMesg">
        </variable>
        <variable name="reservationStatus"
                  messageType="ns0:reservationStatus">
        </variable>
        <variable name="reserveAirlineVar"
                  messageType="ns1:AirlineReservation">
        </variable>
        <variable name="airlineStatusVar"
                  messageType="ns1:AirlineReservationStatus">
        </variable>
        <variable name="reserveCarVar"
                  messageType="ns2:CarReservation">
        </variable>
        <variable name="carStatusVar"
                  messageType="ns2:CarReservationStatus">
        </variable>
        <variable name="reserveHotelVar"
                  messageType="ns3:HotelReservation">
        </variable>
        <variable name="hotelStatusVar"
                  messageType="ns3:HotelReservationStatus">
        </variable>
        <variable name="tempCarStatusVar"
                  messageType="ns2:CarReservationStatus">
        </variable>
        <variable name="tempAirlineStatusVar"
                  messageType="ns1:AirlineReservationStatus">
        </variable>
        <variable name="tempHotelStatusVar"
                  messageType="ns3:HotelReservationStatus">
        </variable>
</variables>
```

Koodiesimerkki 3. Muuttujien määrittäminen prosessissa ReservationSystem

3.4.5 Aktiviteetit

WS-BPEL-aktiviteetit muodostavat prosessin varsinaisen logiikan. Aktiviteetit jaetaan kahteen luokkaan, jotka ovat perus ja rakenteiset. Perusaktiviteetit kuvaavat prosessin etenemisen askeleittain. Rakenteiset aktiviteetit lisäävät hallintarakenteita, jotka voivat sisältää muita peruselementtejä ja/tai raken-

teisia elementtejä rekursiivisesti. WS-BPEL 2.0:n perusaktiviteetit on kuvattu liitteessä 4. WS-BPEL 2.0:n rakenteiset aktiviteetit on kuvattu liitteessä 5.

3.4.6 Näkyvyysalueet

Näkyvyysalueella tarkoitetaan scope-aktiviteetilla rajattua liiketoimintaprosessin osaa, mikä määrittelee sisältämilleen aktiviteeteille niiden suoritukseen vaikuttavan kontekstin. Konteksti voi aktiviteettien lisäksi sisältää variable-, partnerLink-, messageExchange-, correlationSet-, eventHandler-, faultHandler-, compensationHandler- ja terminationHandler-elementtejä. Ylimmän tason näkyvyysalue on process-elementti, minkä sisälle voi scope-aktiviteettien avulla luoda hierarkkisesti useita sisäkkäisiä näkyvyysalueita. Process-elementti eroaa scope-aktiviteetistä kuitenkin siten, että sen sisään ei voi laittaa compensationHandler- eikä terminationHandler-käsittelijöitä.

ReservationSystem-prosessi (liite 2) on liiketoimintaprosessi, jossa ei ole sisäkkäisiä näkyvyysalueita, vaan kaikki käytössä olevat elementit kuuluvat samaan kontekstiin.

3.4.7 Korrelaatio

WS-BPEL-prosessin rajapintakuvauksen mukaan liiketoimintaprosessille tulevien viestien osoite on vastaanottavan palvelun WSDL-portti. Tämä on kuitenkin illuusio, sillä tilallisista WS-BPEL-prosesseista luodaan transaktion alussa aina uusi esiintymä, jotta prosessi voisi palvella jokaista asiakastaan oikein. Siksi prosessille välitettävät viestit tulee osoittaa, ei ainoastaan oikeaan kohdeporttiin, vaan myös prosessin oikeaan esiintymään. [16.]

WS-BPEL -prosessin esiintymän luonti tapahtuu implisiittisesti aloitusaktiviteetin kohdalla. Aloitusaktiviteetti on joko receive- tai pick-aktiviteetti, jossa on asetettu attribuutin createInstance arvoksi ”yes”. [16.]

Jotta prosessin asiakas pystyy kommunikoimaan prosessin oikean esiintymän kanssa, täytyy palvelun välittää asiakkaalle esiintymän yksilöivä tunniste. Tunniste voi sijaita joko viestin kääreessä (envelope), otsikossa (header) tai varsinaisessa dokumentissa. Koska tunnisteen sijainti ei ole riippuvainen esiintymästä, voidaan se ilmaista prosessin kuvauksessa. [16.]

Koodiesimerkissä 4 on esitetty ReservationSystem-prosessin korrelaation käyttöä. Ensin on määritelty correlationSets-elementissä korrelaatiossa käy-

tettävän tunnisteen tyyppi. Samalla kun liiketoimintaprosessi aktivoituu sisään tulevan viestin ansiosta, alustetaan käytettävä correlationSet receive-aktiviteetin correlations-elementissä. Tämän jälkeen WS-BPEL-moottori osaa ohjata kaikki sisääntulevat viestit oikealle prosessi-ilmentymälle kyseisen tunnisteen avulla.

```
<correlationSets>
        <correlationSet name="reservationCorrelationSet"
                properties="ns0:reservationID " />
</correlationSets>

<receive name="reserve" partnerLink="reservationPLink"
        portType="ns0:doReservePortType" operation="reserve"
        variable="reserve" createInstance="yes">
        <correlations>
                <correlation set="reservationCorrelationSet"
                        initiate="yes" />
        </correlations>
</receive>
```

Koodiesimerkki 4. Korrelaation käyttö ReservationSystem-prosessissa

3.4.8 Virheiden käsittely

Liiketoimintaprosessit ovat usein pitkäkestoisia, ja ne voivat käsitellä tietoa lukusissa eri kohteissa. Tällaisessa ympäristössä virheiden käsittely on hyvin vaikeaa ja kriittistä. Tietokantaoperaatioiden ACID-transaktiot (*Atomicity* = jakamattomuus eli kaikkien tapahtumien on onnistuttava tai muuten tapahtuma perutaan, *Consistency* = eheys eli tietokanta vaihtaa tilaa transaktion onnistuessa, *Isolation* = eristyvyys eli transaktiot toimivat toisistaan riippumatta, *Durability* = pysyvyys eli loppuun saatettu transaktio säilyy, vaikka järjestelmä kaatuisi) ovat hyvin lyhytkestoisia, joten voi olla, että liiketoimintaprosessi keskeytyy vasta usean eri ACID-transaktion jälkeen. Tällöin keskeneräiseksi jäänyt työ pitää saada peruttua mahdollisimman hyvin.

WS-BPEL:in tarjoama tapa hallita virheitä perustuu kompensaation-, virheen- ja keskeytyksenkäsittelijöihin sekä joukkoon sääntöjä virheenkäsittelyn järjestyksestä. Kaikille edellä mainituille käsittelijöille on olemassa oletustoteutuksensa, jotka suoritetaan, jos käsittelijöitä ei ole eksplisiittisesti määritelty. Lisäksi näkyvyysalueiden aktiviteettien välille on mahdollista luoda riippuvuussuhteita kontrollilinkkien avulla. [16.]

Esimerkissä Blueprint5 (kts. luku 5) ei ole käytetty minkäänlaista virheenkäsittelyä, vaan virhetilanteista selviytyminen ja niistä ilmoittaminen on täysin WS-BPEL-moottorin vastuulla.

Kompensaationkäsittelijät

Jokaiselle näkyvyysalueelle on mahdollista määrittää oma kompensaationkäsittelijänsä, joka pyrkii tekemään käänteisesti näkyvyysalueen aktiviteettien työn. Poikkeuksena tähän menettelyyn on invoke-aktiviteetti, jolle on mahdollista määritellä oma kompensaationkäsittelijänsä. Virheen sattuessa suoritetaan kaikkien keskeytyspisteeseen asti suoritettujen näkyvyysalueiden ja invoke-aktiviteettien kompensaationkäsittelijät.

Kompensaationkäsittelijä määritellään compensationHandler-elementin sisällä. Kun näkyvyysalueen aktiviteetit on saatu suoritettua onnistuneesti, poistaa WS-BPEL-moottori näkyvyysalueelle asennetun virheenkäsittelijän ja asettaa tilalle kompensaationkäsittelijän. Kompensaationkäsittelijän kutsuminen voi tapahtua virhetilanteen lisäksi myös compensate- tai compensateScope-elementtien toimesta. [16.]

Kompensoinnin oletusjärjestys (Default compensation order)

Kompensoinnin oletusjärjestys perustuu seuraavaan kahteen sääntöön:

1. Jos kahden näkyvyysalueen, A:n ja B:n, välillä on riippuvuussuhde, niin että A:n täytyy olla suoritettu ennen B:n suorituksen alkamista, tulee aina suorittaa kompensointi B:lle ensin.

2. Jos kaksi näkyvyysaluetta, A ja B, ovat toistensa vertaisnäkyvyysalueita, eli niillä on sama isäntänäkyvyysalue, ei näillä vertaisnäkyvyysalueilla saa olla riippuvuuskierrosta. Riippuvuuskierros tarkoittaa sitä, että A:n jokin aktiviteetti on riippuvainen B:n jostain aktiviteetista samalla, kun B:n jokin aktiviteetti on riippuvainen A:n jostain aktiviteetista. Riippuvuuskierroksen omaava prosessikuvaus tulee hylätä.

Virheenkäsittelijät

Virheenkäsittelijä on kompensaation ohella toinen WS-BPEL:in tarjoama tapa toipua virhetilanteesta. Myös virheenkäsittelijän tehtävänä on tehdä käänteisesti oman näkyvyysalueensa virhetilanteeseen asti tekemä työ. Toisin kuin kompensaationkäsittelijän, virheenkäsittelijän voi asettaa myös process-elementin näkyvyysalueella. Toiseksi, jos näkyvyysalueella on suoritettu virheenkäsittelijä, ei kompensaatio samalla näkyvyysalueella ole enää mahdollista. Tämän takia kompensaationkäsittelijät käsittelevätkin virhetilanteita,

mitkä eivät ole syntyneet niiden omalla näkyvyysalueella. Virheenkäsittelijä määritellään faultHandlers-elementin sisällä.

Keskeytyksenkäsittelijät

Keskeytyksenkäsittelijät tarjoavat mahdollisuuden hallita pakotettua prosessin keskeytystä. Ensimmäiseksi keskeytyksenkäsittelijä poistaa näkyvyysalueen tapahtumankäsittelijät ja keskeyttää kaikki aktiiviset aktiviteetit. Tämän jälkeen suoritetaan prosessin määrittelemä keskeytyksenkäsittelijä tai oletuskeskeytyksenkäsittelijä. Toisin kuin virheenkäsittelijä, keskeytyksenkäsittelijä ei voi heittää poikkeusta. Keskeytyksenkäsittelijän suorittamat aktiviteetit lisätään terminationHandler-elementin sisälle.

Virheenkäsittelyn kulku

Virheen tapahtuessa ensimmäisenä keskeytetään kaikki sen näkyvyysalueen aktiviteetit, missä virhe on tapahtunut. Jos aktiviteetti on näkyvyysalue, poistetaan sen virheenkäsittelijä ja suoritetaan keskeytyksenkäsittelijä. Keskeytyksenkäsittelijä keskeyttää ensin kaikki aktiviteetit, mitkä eivät ole näkyvyysalueita, satunnaisessa järjestyksessä. Tämän jälkeen se poistaa kaikista keskeneräisistä näkyvyysalueista virheenkäsittelijät ja käynnistää niiden keskeytyksenkäsittelijät satunnaisessa järjestyksessä. Lopuksi keskeytyksenkäsittelijä käynnistää suoritettujen näkyvyysalueiden kompensaationkäsittelijät noudattaen kompensaation oletusjärjestystä.

Kun keskeytykset on suoritettu, käynnistetään näkyvyysalueen virheenkäsittelijä. Jos eksplisiittistä virheenkäsittelijää ei ole asetettu, käynnistetään oletusvirheenkäsittelijä. Oletusvirheenkäsittelijä suorittaa compensate-aktiviteetin, ja heittää sitten virheen ylemmän tason näkyvyysalueelle.

Oletuskompensaationkäsittelijä ajaa compensate-aktiviteetin, mikä aiheuttaa kaikkien oman näkyvyysalueensa sisältämien näkyvyysalueiden kompensaatioiden kutsumisen kompensaation oletusjärjestystä noudattaen.

On syytä huomioida, että edellä kuvattu virheen-, kompensaation- ja keskeytyksenkäsittely suorittaa keskeytyksenhallinnan keskeneräisille näkyvyysalueille ennen valmiiksi suoritettuja näkyvyysalueita. Keskeneräisen näkyvyysalueen keskeyttäminen tapahtuu satunnaisessa järjestyksessä, vaikka ne olisi liitetty toisiinsa kontrollilinkillä. Valmiiksi suoritetun näkyvyysalueen

kompensoinnissa WS-BPEL taas huomioi kontrollilinkkien asettamat keskinäiset riippuvuudet.

Virheiden käsittelyn monimutkaisuuden takia prosessin tekijän tai testaajan voi olla hyvinkin vaikeaa hahmottaa etukäteen, miten kompensaationkäsittely etenee virheen sattuessa. Tämän takia esimerkiksi IBM on tehnyt esityksen kompensaationkäsittelyn yksinkertaistamisesta esim. oletuskäsittelijät kokonaan poistamalla. [22.]

3.4.9 Tapahtumien käsittely

Jokaisella näkyvyysalueella, mukaan lukien process-elementin näkyvyysalue, voi olla omat tapahtumankäsittelijänsä. Ne määritellään eventHandlers-elementin sisällä. Niitä voidaan suorittaa samanaikaisesti ja ne käynnistyvät kyseisen tapahtuman tapahtuessa. Tapahtumankäsittelijän lapsiaktiviteetin tulee olla aina scope-aktiviteetti, eli näkyvyysalue, mikä voi sisältää muita aktiviteetteja.

Tapahtumia on kahdentyyppisiä, sisään tulevia viestejä ja hälytyksiä. Hälytys tapahtuu yleensä ennalta asetetun ajan kuluttua, tai tiettyyn ajanhetkeen saavuttaessa. Tapahtumankäsittelijän suoritus on osaa normaalia näkyvyysalueen toimintaa, toisin kuin virheen-, kompensaation- tai keskeytyksenkäsittelijöiden.

Blueprint5-esimerkissä ei ole käytetty tapahtumankäsittelijöitä, vaan sisään tuleviin viesteihin reagoidaan ainoastaan receive- ja pick-aktiviteettien avulla (kts. liite 2). Samoin kuin eventHandlers-elementti, pick-aktiviteetti pitää sisällään joukon tapahtumia. Eroja tapahtumankäsittelijän ja pick-aktiviteetin välillä on kuitenkin useita. Tapahtumankäsittelijän sisältämä tapahtuma voi aktivoitua milloin tahansa omalla näkyvyysalueellaan. Pick-aktiviteetin sisältämä tapahtuma voi aktivoitua vain, kun kyseistä pick-aktiviteettia suoritetaan. Tapahtumankäsittelijän tapahtumat voivat myös olla suorituksessa samanaikaisesti ja voivat aktivoitua, vaikka näkyvyysalueella olisi suorituksessa keskeneräisiä aktiviteetteja.

3.5 BPMN

Business Process Modeling Notation (BPMN) on BPMI:n kehittämä standardi liiketoimintaprosessien mallinnukseen. Se oli BPML-kielen tytärprojekti, ja määritteli liiketoimintaprosessin graafisen notaation tavalla, mikä muistuttaa

paljon UML:n aktiviteettikaavioita. BPMN kehitys jatkuu nykyään OMG:n alla, ja sen nykyinen versio on BPMN 2.0 [23].

BPMN sisältää BPEL-kartoituksen tarkoittaen, että sitä voidaan käyttää määrittelemään BPEL-prosesseja. BPEL onkin käytännössä korvannut BPML:n suorituskielenä, ja koska BPEL ei ota mitenkään kantaa graafiseen notaatioon, tuo BPMN sille tervetulleen standardinotaation.

BPMN:n käyttäminen BPEL:illä suoritettavien liiketoimintaprosessien kuvaukseen on ideana hieno, mutta käytännössä on esiin noussut monia ongelmakohtia. Jotta liiketoiminta-analyytikot (eng. business analyst) pystyisivät helposti BPMN:n avulla luomaan suoritettavia prosesseja, edellyttää se 300-sivuisen BPMN-spesifikaation tuntemista ja omaksumista, mikä taas tuottaa suuria koulutuskustannuksia yrityksille. Toiseksi, suoritettavan BPMN-prosessin suunnittelu vaatii ajattelua järjestelmän näkökulmasta, kun taas liiketoiminta-analyytikot toimivat normaalisti asiakkaan näkökulmasta. Tämän takia tarvitaan yleensä välikäsi, joka muuntaa liiketoiminnallisen prosessin suoritettavaksi prosessiksi. [24.]

BPMN on myös ominaisuuksiltaan laajempi kuin BPEL, joten kaikki BPMN:n ominaisuudet eivät suoraan käänny WS-BPEL-muotoisiksi. Tämä on aiheuttanut pitkään ongelmia prosessinmallinnusvälineille, jotka vastaavat BPMN-prosessin kääntämisestä suoritettavaksi WS-BPEL-prosessiksi. Pitkään WS-BPEL-prosessit eivät kääntyneet BPMN-kaavioiksi eli käännös oli yhdensuuntainen. Tästä johtuen esimerkiksi suosittu IDE, NetBeans, on tehnyt omassa Orchestration Designerissaan pieniä kompromisseja BPMN:n käytön suhteen, ja näin saanut myös BPMN-BPEL-BPMN -kiertomatkan toimimaan [25]. Active Endpoints ActiveVOS:n ja eClarus Business Process Modeler:in uusimmat versiot näyttävät kuitenkin ratkaisseen kiertomatkaongelman.

Eclipsen SOA Tools Platform (STP) -projektin STP Intermediate Model (STP-IM) -aliprojektin tarkoituksena on luoda silta eri STP-editorien, (mm. BPMN ja BPEL) välille yleisen mallin avulla. Tämä mahdollistaa sen, että liiketoiminta-analyytikot voivat suunnitella prosessin BPMN-editorilla, minkä jälkeen ohjelmistoasiantuntijat voivat jatkaa prosessin kehittämistä BPEL-editorissa. Tämä menettelytapa onkin ehkä suositeltavin, jotta sekä liiketoiminta-analyytikot että ohjelmistoasiantuntijat pysyvät tyytyväisinä.

3.6 WS-BPEL ja EAI

Perinteiset integraatiomenetelmät ja liiketoimintaprosessien automatisointiin tarkoitetut järjestelmät sisältävät tyypillisesti logiikkaa tiettyihin liiketoiminta-tarkoituksiin kuten toiminnanohjaukseen (ERP) tai asiakkuudenhallintaan (CRM). Jos näihin sovelluksiin tarvitsee tehdä muutoksia, niiden kehittämiseen, testaamiseen ja käyttöönottoon laitetun panostusten suuruus tekee integroinnista ja prosessinmuutoksista sekä kalliita että monimutkaisia. [3.]

WS-BPEL ja Web Services tarjoavat standardoidun integraatiorajapinnan ja standardoidun kielen integrointiin ja prosessien automatisointiin. Michael Cobban, SoftCare-yhtiön johtava liiketoiminnallinen analyytikko, perustelee artikkelissaan ”Mikä on BPEL ja miksi se on niin tärkeä yritykselleni?” WS-BPEL:in valitsemista seuraavasti:

> ”WS-BPEL:illä on jopa potentiaalia tehdä EAI- ja BPM-ratkaisujen ominaisuuksista massahyödykkeitä edullisesti. WS-BPEL:iin verrattuna EAI ja BPM-rakaisut ovat monesti yksinkertaisesti vain liian kalliita. Ne ovat myös kalliita kehittää, ottaa käyttöön, ylläpitää ja levittää moninaisiin liiketoimintajärjestelmiin koko yrityksen laajuisesti. Näiden järjestelmien tukemiseen tarvittavat erityistaidot luovat kustannuksensa ja saatavuusongelmansa.” [3.]

WS-BPEL on erityisen käyttökelpoinen ympäristöissä, joilla on jo olemassa useita Web Services -rajapintoja. Mitä suurempi määrä palveluja on tarjolla, sitä kallisarvoisemmaksi BPEL tulee. Integroinnin helppous on syy siihen että Web Services on tullut yhdeksi IT-maailman kuumimmista teknologioista. [3.]

3.7 WS-BPEL ja ESB

On olemassa myös muita kuin Web Services -teknologiaan pohjautuvia SOA-toteutuksia, jotka tarjoavat samanlaisia etuja. Ideaalisten SOA-palveluiden tulisi olla protokollariippumattomia tarkoittaen sitä, että eri käyttäjät voivat kommunikoida palvelun kanssa eri tavoin. Kuluttajien ja palvelun-tarjoajien välissä tulisi aina olla hallinnointikerros, mikä takaisi täydellisen joustavuuden käytettävien protokollien osalta. Tähän ongelmaan eräs ratkaisu on ESB eli Enterprise Service Bus.

ESB toimii nimensä mukaisesti väylänä, jota pitkin siihen kytkeytyneet palvelut voivat kommunikoida keskenään. Palvelut voivat käyttää viestien lähet-

tämiseen mitä tahansa ESB:n tukemaa protokollaa ja viestityyppiä, sillä ESB hoitaa viestien välittämisen ja muuntamisen palveluiden välillä. Esimerkiksi, kun jokin palvelu lähettää ESB:n välityksellä JMS-muotoisen (Java Message Service) viestin WS-BPEL -prosessille, ESB muuntaa viestin SOAP-muotoon. Näin voidaan todeta, että WS-BPEL ja ESB eivät suinkaan ole vaihtoehtoisia tapoja toteuttaa SOA, vaan ne voivat toimia käsi kädessä kun halutaan tarjota WS-BPEL-palveluja myös muille kuin Web Services -teknologiaa käyttäville sovelluksille.

3.8 WS-BPEL-tuotteet ja graafiset työkalut

WS-BPEL-moottoreiden valmistajia ja tuotteita on paljon. Taulukko 2 on lueteltu joitakin tunnetuimpia.

Taulukko 2. WS-BPEL-moottoreita

Tuote	Valmistaja	Kehys	Lisenssi
ActiveVOS	Active Endpoints	Servlet/ JavaEE	Kaupallinen
Apache ODE	Apache Software Foundation	Apache Axis/ Servicemix	Apache
Biztalk Server	Microsoft	.NET	Kaupallinen
jBPM	jBoss	JavaEE	LGPL
Open ESB	Oracle (Sun)	JavaEE / JBI	Open Source, CDDL
WebSphere Process Server	IBM	JavaEE	Kaupallinen

Kaupallisiin versioihin kuuluvat yleensä myös graafiset hallinnointi- ja testaustyökalut.

Liiketoimintaprosessien graafiseen mallintamiseen tarkoitettuja BPMN-tuotteita on tällä hetkellä yli 60. Ne löytyvät OMG:n nettisivulta http://www.bpmn.org/BPMN_Supporters.htm. Niiden lisäksi myös NetBeans- ja Eclipse-sovelluskehittimiin on saatavilla BPMN-plugineja.

4 KÄYTTÖTAPAUKSIA

WS-BPEL:iä voidaan käyttää moneen eri tarkoitukseen. Seuraavaksi käydään kolmen esimerkin kautta läpi, kuinka eräät yritykset ovat ottaneet BPEL:in osaksi integrointiratkaisujaan.

4.1 Kiinteistönvälitysyritys Move Inc. [26.]

Move Inc:in verkkopalvelu move.com on tarkoitettu kaikille kodin ostamisesta tai vuokrauksesta kiinnostuneille tarjoten myös logistiikka- ja muuttopalveluita. Move Inc. on laajentanut toimintojaan paljon yritysfuusioilla ja muilla hankinnoilla ja tuonut näin yhteen useita eri Web Services -sivustoja.

Mike Remedios on Move Inc:in teknologiajohtaja. Hänen vastuullaan on lukuisten yrityskauppojen kautta hankittujen sovellusten integrointi, jotta ne tarjoaisivat yhtenäisen näkymän kaikesta niiden sisältämästä tiedosta asiakkaille.

Remedios on oman määritelmänsä mukaisesti keskittynyt ”käytännölliseen SOA:an”. Hän tarkoittaa sillä asteittaista käytössä olevien vanhojen sovellusten integroimista. Pitkän tähtäimen tavoitteena on tuoda tarvittavaa ketteryyttä online-liiketoimintaan, jotta uudet innovaatiot voidaan nopeasti kytkeä Web Services -sivustoihin.

> ”Me olemme osittain SOA. Kuten lukuisat muut yritykset, me olemme saaneet muista ostamistamme yrityksestä koodispagettia, jota pilkomme rakeisuudeltaan vaihteleviin osiin SOA:a varten.”

Näin hän pyrkii välttämään sudenkuoppia ja IT-resurssien ylikuormittamista massiivisella projektilla, joka tuottaisi vain vähän välitöntä voittoa investoinnille. Hän ei halua projektinsa olevan yksi niistä massiivisista projekteista, joilla on vähän liiketoiminnallista arvoa lyhyellä tähtäimellä ja jotka syövät resursseja.

Remedios käyttää Oracle BPEL Process Manager:ia asiakastietojen integroimiseen tuottaakseen yksittäisen näkymän asiakkailleen ja henkilökunnalle tarjoamistaan palveluista. Se käyttää myös apunaan Siebel-asiakkuudenhallinta- ja PeopleSoft-toiminnanhallintajärjestelmiä.

Move Inc:in täytyy integroida vanhat sovellukset uusiin niin, että ne voivat kommunikoida keskenään ja liittyä käyttöliittymillä puhelinkeskukseen ja Web Services -sivustoihin. Myös automaattiset ja manuaaliset liiketoiminta-prosessit tulee integroida yhdessä sovelluksien kanssa. Tässä BPEL tulee mukaan kuvioihin.

Move Inc. käyttää BPEL:iä tällä hetkellä perinteiseen EAI-tyyliin SOA:n ensimmäisenä askeleena. Päämääränä heillä on luoda myyntihenkilöstölle yhtenäinen näkymä potentiaalisesta asiakkaasta aina laskutukseen asti.

Move Inc. käyttää BPEL:iä käsittelemään ”pitkäkestoisia transaktioita”. Kiinteistönvälityksessä esimerkiksi kodin ostaminen voi viedä 30 päivää mennäkseen prosessin läpi. Tapauksissa, joissa työnohjausjärjestelmä ei pysty käsittelemään pitkäkestoisia transaktioita, lähetetään viesti Oracle BPEL Process Managerille, joka on linkitetty pääviestiväylään. BPEL-moottori sitten monitoroi transaktiota ja varmistaa, että se valmistuu ja että palvelutasovaatimukset täytetään.

> ”Vaikka BPEL- ja SOA-toteutukset usein saavat hyväksyntänsä johtajilta kustannussäästöjen perusteella, ne ovat paljon arvokkaampia pitkällä tähtäimellä, sillä ne mahdollistavat uudet liiketoiminnalliset innovaatiot kuten päivitykset Web-sovelluksiin. Tällä tavalla BPEL on paljon enemmän strateginen kuin taktinen työkalu.”

4.2 Verizon Wireless [27.]

Verizon Wireless on mobiiliverkkoihin erikoistunut teleoperaattori. He käyttävät Oracle BPEL Process Manager:in orkestroimaa SOA-sovellusta huijausyritysten selvittämisessä. Jan Shook, pääarkkitehti petostentorjuntatiimissä, toteaa J2EE-koodaajien saattavan olevan pettyneitä, sillä hän suunnitteli uudelleen koko huijausyritystenetsintäsovelluksen Oracle BPEL:illä. Java-koodia ei siis ole lainkaan jäljellä.

Kun Jan Shook aloitti pääarkkitehtina, käytössä oli J2EE-sovellus kolmella isolla tietokannalla. Järjestelmä käsitteli 2,5 miljardia tietuetta päivässä, ja 100 henkilön tiimin täytyi käydä tietomassa lävitse etsien poikkeavuuksia, jotka saattaisivat tarkoittaa petoksellista kännykänkäyttöä.

> “Me rakensimme sovelluksen BPEL-pohjaisena ratkaisuna, joten meillä ei todella ole yhtään J2EE-tavaraa enää. Päädyimme puhtaaseen BPEL-ratkaisuun yhdellä Shockwave-

> tiedostolla ja yhdellä JSP-sivulla. Adobe Flex:iin ja Flashiin perustuva web-käyttöliittymä tekee Web Services -palvelukutsuja BPEL:iin, ja käytämme BPEL:iä välittämään prosesseja. Se käynnistyy ja tekee kaiken orkestroinnin yhdessä muiden yritysjärjestelmien kanssa."

Kuuden muun henkilön kanssa Jan Shook kirjoitti tietokantapuolelle liiketoimintasäännöt, jotka suodattavat kahden ja puolen miljardin tietueen joukosta ne, joilla on epäilyttäviä ominaisuuksia. Sitten ne välitetään BPEL-moottorille, joka käsittelee epäilyttävät tapaukset.

Dave Chappell, Oraclen varapääjohtaja ja SOA-teknologiavastaava, kirjoitti blogissaan uudesta sovelluksesta näin:

> "Usein sanotaan että paras rivi koodia on se jota ei koskaan tarvitse kirjoittaa. Tämä uusi toteutus on 0,5% alkuperäisestä koostaan. Tämä on suoraa seurausta Oracle BPEL Process Managerin ja sääntömoottorin käytöstä räätälöidyn koodin sijasta."

Koodin vähentämisen lisäksi uusi toteutus on myös dramaattisesti vähentänyt käytettävän laitteiston määrää ja siten myös energiankulutusta. Chapellin mukaan BPEL-pohjainen SOA-toteutus eliminoi kuusi E-luokan Sun Microsystems:in palvelinta 192-prosessoreilla ja korvasi ne yhdellä kahdeksanytimisellä Sun UltraSPARC T1:llä, mikä käyttää Niagara-prosessoriarkkitehtuuria. Tietokantojen tallennuskapasiteettivaatimukset ovat vähentyneet 20+ teratavusta 64 gigatavuun. Chappell arvioi muutosten ohjelmistoissa ja laitteistoissa vähentäneen energiankulutusta jopa 99,5 %.

WS-BPEL-järjestelmä mahdollistaa myös sen, että Verizon pääsee käsiksi tietoihinsa milloin vain. Tämä on hyvin merkittävää, sillä heidän ei tarvitse enää replikoida tietokantojaan, vaan Verizon Wireless voi tehdä hakuja suoraan ulkoisista järjestelmistä.

Lisäpalveluita hyväkseen käyttämällä tieto saatetaan oikeaan muotoon. Näitä tietoja tarvitaan tärkeiden päätöksien tekemiseen petosyrityksiin liittyen ja tuottamaan yksityiskohtaisia raportteja havaituista liiketoimintapoikkeuksista.

4.3 Sovelluspalveluiden tarjoaja USinternetworking Inc. [28.]

AT&T:n ostama USinternetworking Inc (USi) on sovelluspalveluiden tarjoaja, mikä isännöi ERP-sovelluksia, joita ovat mm. Oracle Application Suite, PeopleSoft ja Siebel. Sen lisäksi, että he tarjoavat tietokeskuksen ja laitteis-

ton asiakkailleen, USi:n tarjoama lisäarvo on sovellusten hallinnassa ja tukipalveluissa.

Kun he alkoivat tutkia tuottamiaan prosesseja, he huomasivat, että ne eivät olleet enää kehittyvien tarpeiden mukaisesti ajan tasalla. Esimerkkinä teknisen varapääjohtaja Mike Rulf antaa tapauksen, jossa täytyy varata IP-osoite hallinnoitavalle palvelimelle.

> ”Ennen vanhaan IP-osoitteen varausprosessi tapahtui siten että varausvastaavan täytyi kävellä verkkopalveluosastolle ja sanoa, ’Voisitteko varata IP-osoitteen näille kolmelle palvelimelle joita ylläpidämme tälle asiakkaalle.’ Verkkopalveluvastaava sitten vastasi: ’Okei, laitan sen Saapuneet-laatikkooni ja palaan asiaan muutaman tunnin päästä.’ Kun varausvastaava sitten lähti huoneesta, verkkopalveluvastaava kirjoitti Perl-skriptin, ajoi sen varatakseen IP-osoitteen, haki kupin kahvia ja teki muutaman muun operaation ennen kuin täytti tarvittavan lomakkeen ja toimitti sen varausvastaavalle.”

Tämä ei tietenkään ollut kovin tehokasta, ja tarvittava paperityön määrä on suuri. Tämä motivoi tietenkin Rulfia korvaamaan traditionaalisen prosessin yhdistelemällä SOA:a, Web Services -palveluita ja BPEL:iä.

Konversion alussa Rulf ja hänen tiiminsä katsoivat, mitä vanhasta Perl-sovelluksiin perustuvasta hallinnointijärjestelmästä voitaisiin automatisoida.

> ”Kävi ilmi, että suurelle osalle prosessimme askelista oli olemassa pieniä perl-skriptejä ja erilaisia paloja koodia, jotka sisälsivät hieman dokumentaatiota, joiden avulla saatoimme tarkistaa prosessin. Päätimme, että meidän tulisi kääriä nuo koodin palaset ja muuntaa ne Web Service:iksi.”

> ”Niinpä siis otin kääreen ja standardoin sille tietoturvamallin, sillä se on hyvin tärkeää, jotta voidaan tarkistaa, että se, joka teki varauksen, omisti todella oikeudet tehdä sen. Käärin mukaan myös vähän standardivirheenkäsittelyä, jotta saavutetaan tunnetut poikkeustilanteet, joita vasten voidaan toimia. Sitten otin nämä koodin palaset ja laitoin ne kääreen sisään, ja näin minulla oli Web Service, joka tekee (IP-osoitteen) varauksen, noudattaa tunnistussääntöjä ja jota BPEL-prosessi voi kutsua. Tämä tekee siitä hyvin valvottavan ja hallittavan prosessin.”

BPEL-prosessi on myös sähköisesti jäljitettävä. Silloin tällöin Rulfin luona vierailee tarkastajia, jotka ovat kiinnostuneita tiettyjen asiakkaiden palveluista. Kun he kysyvät esimerkiksi, kuinka IP-osoitteen varaus tehtiin tietylle asiakkaalle, Rulf näyttää heille BPEL-moottorin prosessin.

> ”Se näyttää prosessin kulun, kuka teki mitä ja milloin, kuinka tieto muuttui, ja mitä tuloksia saatiin koko prosessin kuluessa. Heille (tarkastajille) on paljon nopeampaa katsoa diagrammia kuin käydä läpi paksuja paperinippuja. Tämä on paljon taloudellisempaa, sillä tarkastajat viipyvät lyhyempään ja varausvastaavan roolista on voitu luopua.”

Rulf arvioi, että uuden WS-BPEL-sovelluksen aikaansaamat kustannussäästöt voivat olla jopa enemmän kuin 15 %.

4.4 Yhteenveto käyttötapauksista

Edelläolevissa esimerkeissä WS-BPEL:iä käytettiin seuraaviin käyttötarkoituksiin:

1. Legacy-järjestelmien integrointi
2. Java EE -sovelluksen korvaaminen uudella WS-BPEL-toteutuksella ja tietokantasäännöillä
3. Manuaalisen prosessin korvaaminen.

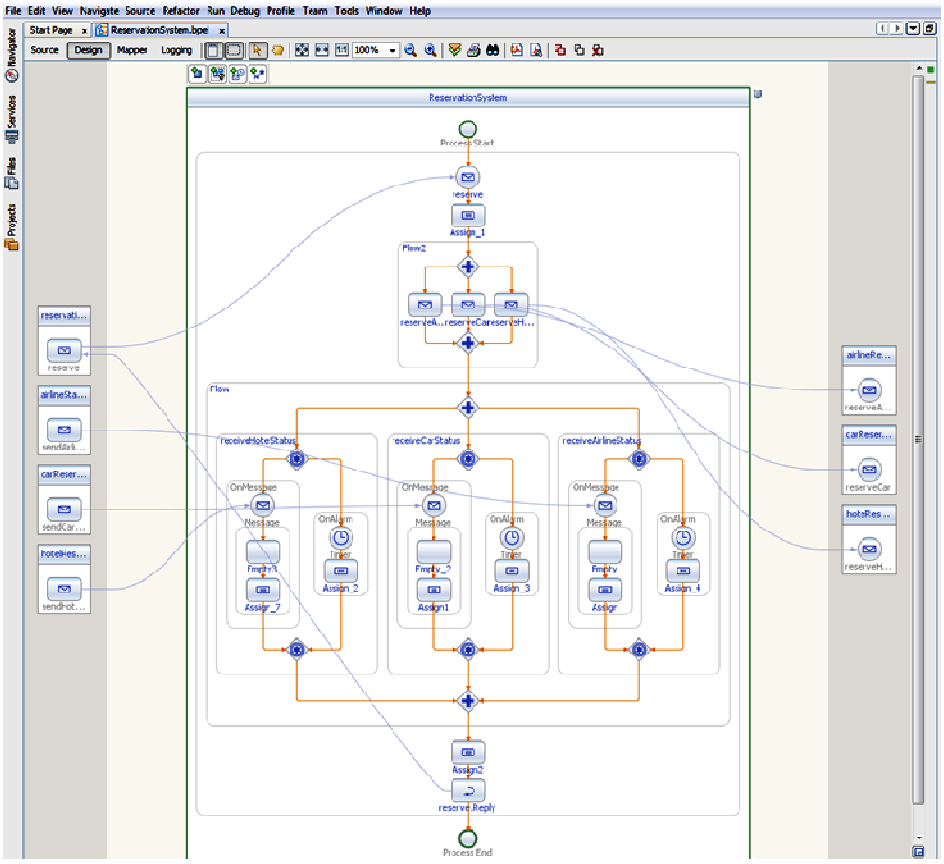
5 WS-BPEL-PROSESSIEN SIIRRETTÄVYYS

Eräs Web Services -standardin tuomista eduista on valmistajariippumattomuus. Sovellukset pystyvät integroitumaan keskenään SOAP-rajapinnan yli, vaikka rajapinnan toteuttama teknologia olisi eri valmistajalta. Kuten aikaisemmin mainittiin, voidaan WS-BPEL:iä pitää yhtenä Web Services -teknologian laajennuksena. Mutta kuinka uskollisia eri valmistajien toteuttamat WS-BPEL-moottorit ovat OASIS:n standardille? Kuinka helppoa on vaihtaa WS-BPEL-tuotetta eli siirtää olemassa olevat prosessit jonkin toisen valmistajan WS-BPEL-moottorille? Nämä olivat tämän työn tilaajan keskeisimmät kysymykset.

Kun Internetistä etsittiin tietoa eri WS-BPEL-moottoreista, tuli nopeasti ilmi, että eri valmistajat ovat kehittäneet tuotteisiinsa laajennuksia, jotka eivät kuulu WS-BPEL 2.0 -standardiin. Ongelmaa päätettiin lähestyä käytännön kokemusten pohjalta valitsemalla kaksi vapaassa jakelussa olevaa WS-BPEL-moottoria ja siirtämällä toiselle kehitetty prosessi toiseen. Valituiksi tulivat Sun Microsystems:in (nyk. Oracle) GlassFish ESB sekä Apachen ODE. GlassFish ESB on Oraclen kaupallisesti tukema versio OpenESB:stä, joka pitää sisällään myös WS-BPEL-moottorin. Apache ODE puolestaan on puhdasverinen WS-BPEL-moottori. Molempia yhdistää se, että ne ovat avoimen lähdekoodin Java-projekteja.

5.1 GlassFish ESB

GlassFish ESB on Open ESB:n kaupallisesti tuettu versio. Tähän työhön valittiin uusin versio 2.2, joka julkaistiin joulukuussa 2009. GlassFish ESB 2.2:n mukana tulee NetBeans:in versio 6.7.1, jossa on BPMN-notaatiota noudattava editori WS-BPEL-prosesseille. Mukana tulee myös kahdeksan hyvää esimerkkitapausta, joista yksi valittiin tämän työn siirrettäväksi prosessiksi. Kuvassa 5 on kuvakaappaus NetBeansin BPMN-editorista, jossa käsitellään luvussa 5.3 esiteltävää ReservationSystem-prosessia.



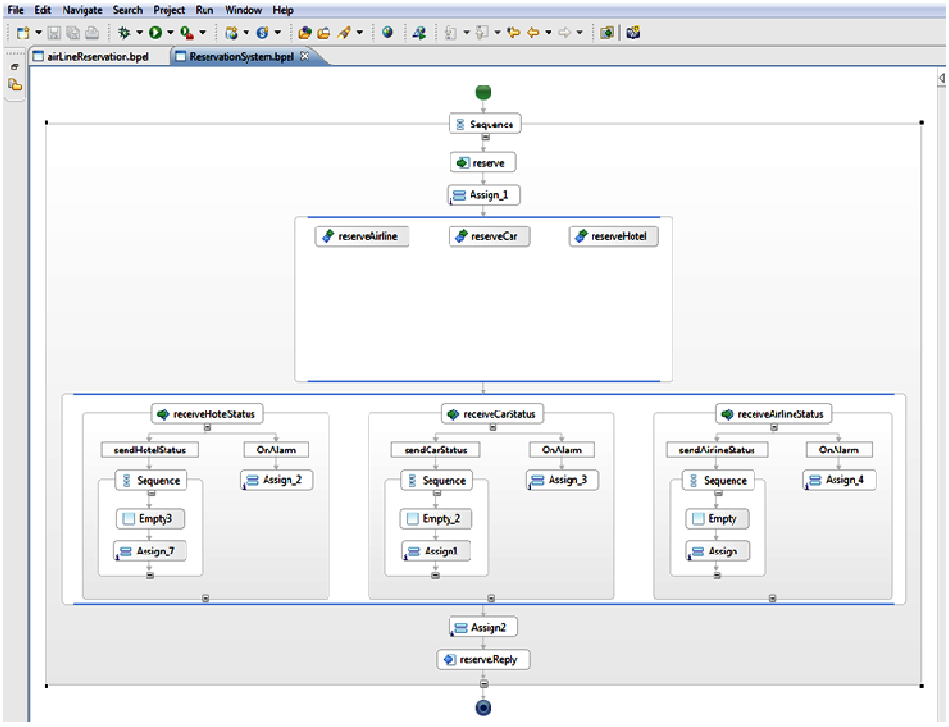
Kuva 5. NetBeans 6.7.1 BPEL Designer ja ReservationSystem-prosessi

5.2 Apache ODE

Apache ODE:sta on saatavissa sekä Axis2 että JBI-versiot. Axis2-version voi kytkeä mihin tahansa J2EE-standardin mukaiseen web-säiliöön (container). JBI-version puolestaan pystyy liittämään ainakin Apache Servicemixiin, mikä on Apachen oma ESB-tuote. Tähän työhön valittiin kesäkuussa 2010 julkaistu 1.3.4-versio Axis2:lle.

Apache ODE:n mukana ei tule graafista editoria, mutta pienellä selvityksellä saatiin selville, että Eclipselle on olemassa BPEL Designer -plugin, jossa on tuki Apache ODE:lle. BPEL Designer käyttää omaa notaatiotaan, ei BPMN:ää. Eclipselle on olemassa myös SOA Tools -projektiin kuuluva BPMN Modeler, jolla tehtyjä prosesseja voi muuntaa WS-BPEL-muotoon. Lyhyen tutustumisen jälkeen päädyttiin kuitenkin valitsemaan BPEL Designer sen helppokäyttöisyyden takia. Kuvassa 6 on kuvakaappaus BPEL Designer -editorista, jossa käsitellään luvussa 5.3 esiteltävää ReservationSystem-prosessia.

Myös Apache ODE:lle on maksullista tukea saatavilla, nettisivujen mukaan tällä hetkellä ainoa yritys on tosin puolalainen Touk (www.touk.pl).



Kuva 6. Eclipse 3.6 BPEL Designer ja ReservationSystem-prosessi

5.3 Siirrettävä prosessi

Tämän työn siirrettäväksi prosessiksi valittiin eräs GlassFish ESB:n mukana tulleista esimerkkiprosesseista, nimeltään BluePrint5. Tämä prosessi päätettiin valita siksi, että se käytti riittävän monipuolisesti WS-BPEL:in ominaisuuksia ja Sunin (nykyään Oraclen) tukemana kehitysprojektina GlassFish ESB:lle tehdyn prosessin oletettiin olevan riittävästi WS-BPEL 2.0 -standardin mukainen.

BluePrint5 toteuttaa yksinkertaisen keskitetyn varausjärjestelmän, jossa yhden palvelun kautta voi varata auton, hotellin ja lentolipun. Toteutus koostuu neljästä WS-BPEL-prosessista:

- ReservationSystem
- AirlineReservation
- CarReservation
- HotelReservation

ReservationSystem ottaa vastaa käyttäjän tilauksen ja lähettää sen samanaikaisesti eteenpäin kolmelle alijärjestelmälle. Kaikkia alijärjestelmiä kutsutaan samanaikaisesti, joten myös vastausten osalta varaudutaan rinnakkaiseen käsittelyyn. Koska ReservationSystem on asynkroninen, varaudutaan vastauksen puuttumiseen asettamalla ajastin, joka lauetessaan mahdollistaa vaihtoehtoisen suorituspolun.

Kolme alijärjestelmää ovat tässä esimerkissä WS-BPEL-prosesseja, mutta ne voisivat aivan yhtä hyvin olla WS-palveluja. Kuten aikaisemmin on mainittu, WS-BPEL-prosessit näkyvät omille asiakkailleen tavallisina WS-palveluina.

AirlineReservation-, CarReservation-, ja HotelReservation-prosessit tutkivat ReservationSystem-prosessin lähettämää varausta. Jos varausviestistä löytyy kunkin alijärjestelmän etsimä tieto, lähetetään ReservationSystem-prosessille tieto onnistuneesta varauksesta. Alijärjestelmät ovat niin kutsuttuja tynkiä (stub), eli ne eivät sisällä oikeata varausjärjestelmää vaan ainoastaan sen toimintaa imitoivan tyngän.

Esimerkin kuvaus löytyy myös osoitteesta:

https://blueprints.dev.java.net/bpcatalog/ee5/soa/BP5_DesignDetails.html

Esimerkin sisältämät tiedostot on esitetty taulukossa 3. Jokaisella WS-BPEL-prosessilla on oma rajapintakuvauksensa erillisessä WSDL-tiedostossaan. Lisäksi pää- ja alijärjestelmien yhteinen tietorakenne on määritelty XML-skeemassa reservation.xsd.

Taulukko 3. BluePrint5:n tiedostot

Tiedosto	**Kuvaus**
airLineReservation.bpel	Aliprosessi
AirLineReservation.wsdl	Aliprosessin rajapintakuvaus
carReservation.bpel	Aliprosessi
CarReservation.wsdl	Aliprosessin rajapintakuvaus
hotelReservation.bpel	Aliprosessi
HotelReservation.wsdl	Aliprosessin rajapintakuvaus
ReservationSystem.bpel	Pääprosessi [LIITE 2]
Reservation.wsdl	Pääprosessin rajapintakuvaus [LIITE1]
reservation.xsd	Yhteinen tietorakenne (XML-skeema)

5.4 Esimerkkiprosessin siirtämisen työvaiheet

Seuraavaksi käydään läpi kaikki työvaiheet BluePrint5-esimerkin siirtämiseksi Apache ODE:lle lähtien tilanteesta, jossa GlassFish ESB on asennettu työasemalle.

5.4.1 BluePrint5-esimerkin testaus GlassFish ESB:ssä

BluePrint5-esimerkin todettiin toimivan GlassFish ESB:ssä suorittamalla esimerkin mukana tulleet testitapaukset. Testitapaukset ja -tulokset on esitetty taulukossa 4.

ReservationSystem-prosessille lähetettävä varausviesti sisältää reservationID-kentän ja description-kentän. ReservationID on kunkin varauksen yksilöivä tunniste. Description pitää sisällään varsinaisen varaustapahtuman kertoen, halutaanko varata lento, auto ja/tai hotelli.

ReservationSystem-prosessi kokoaa lähettämänsä vastausviestin eri alijärjestelmiltä tulevista vastauksista. Alijärjestelmien vastaukset ovat boolean-arvoja kertoen, onnistuiko varaus vai ei.

Taulukko 4. BluePrint5-testitapaukset ja -tulokset GlassFish ESB:llä

Testi	Varausviesti	Vastausviesti
1	reservationID: 0000098762 description: book_airline	airlineStatus: true hotelStatus: false carStatus: false
2	reservationID: 0000098761 description: book_car, book_hotel, book_airline	airlineStatus: true hotelStatus: true carStatus: true
3	reservationID: 0000098762 description: book_hotel	airlineStatus: false hotelStatus: true carStatus: false

5.4.2 Apache ODE:n asentaminen

Apache ODE:n saa ladattua osoitteesta http://ode.apache.org/getting-ode.html. Tätä työtä varten asennettiin WAR-distribuution uusin versio, 1.3.4. WAR-distribuutio tulee asentaa johonkin Java2EE-standardin web-säiliöön, joten sitä varten asennettiin Apache Tomcat:in versio 6.0.30.

Apache ODE:n asennus Tomcatiin tapahtuu kopioimalla ODE:n asennuspaketin juuressa oleva ode.war Tomcatin webapps-hakemistoon. Kun Tomcat

käynnistetään, purkaa se war-tiedoston omaksi hakemistokseen webapps-hakemiston alle.

5.4.3 Eclipse SOA Platform:in asentaminen

BPEL Designer -pluginia varten tulee asentaa Eclipse Helios:n Eclipse IDE for Java EE developers tai Eclipse SOA Platform -versio. Tätä työtä varten valittiin Eclipse SOA Platform, jonka saa ladattua osoitteesta http://www.eclipse.org/downloads/. Asentaminen tapahtuu purkamalla asennuspaketti haluttuun sijaintiin.

5.4.4 BPEL Designer -pluginin asentaminen

BPEL Designer Eclipse-projekti (http://www.eclipse.org/bpel/) on WS-BPEL-prosessien tekemiseen tarkoitettu graafinen, Eclipsen GEF-kirjastoa (Graphical Editing Framework) käyttävä editori.

BPEL Designerin saa asennettua valitsemalla Eclipsen Help-valikosta ”Install New Software”, ja lisäämällä uudeksi asennussijainniksi BPEL Designer Update -sijainti: http://download.eclipse.org/technology/bpel/update-site/. Tarkemmat ohjeet asentamiseen löytyvät osoitteesta http://www.eclipse.org/bpel/install.php.

5.4.5 Apache ODE:n liittäminen Eclipseen

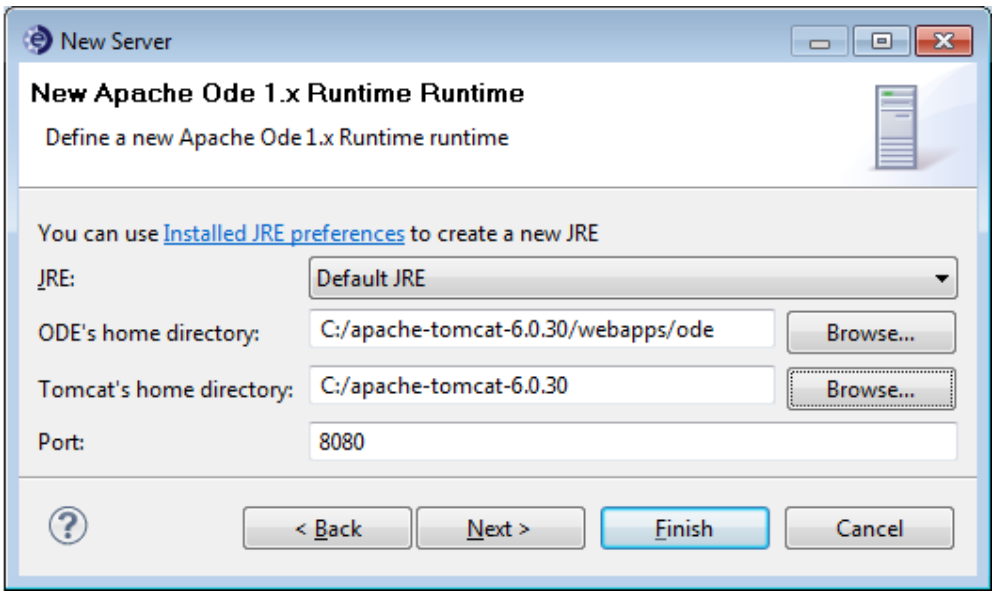
BPEL Designer -pluginin mukana tulee tuki Apache ODE:lle. Sen liittäminen Eclipseen onnistuu seuraavasti:

1. Valitaan Eclipse-perspektiiviksi BPEL-perspektiivi (Window -> Open Perspective -> Other -> BPEL Perspective).

2. Lisätään uusi palvelin Server-näkymään (klikataan hiiren kakkosnäppäimellä Server-näkymää ja valitaan New -> Server).

3. Valitaan Apache-kansiosta ”ODE v1.x Server” ja painetaan ”Next”.

4. Määritellään Apache ODE:n ja Tomcat:in kotikansiot kuvan 7 mukaisesti ja painetaan ”Finish”.



Kuva 7. Apache ODE -runtimen luominen

5.4.6 BPEL-projektin luominen Eclipseen

Seuraavaksi luodaan uusi BPEL-projekti Eclipseen seuraavasti:

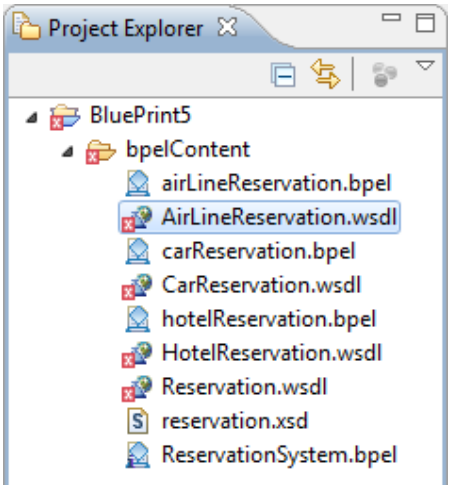
1. Valitaan File-valikosta New -> Project.
2. Valitaan BPEL 2.0 -kansion alta ”BPEL Project” ja painetaan ”Next”.
3. Kirjoitetaan BPEL Project -velhoon projektin nimeksi ”BluePrint5”.
4. Valitaan Target Runtime:ksi aiemmin luotu ”Apache Ode 1.x Runtime” ja painetaan ”Finish”.

5.4.7 BluePrint5-esimerkin importointi Eclipse-projektiin

Seuraavaksi importoidaan BluePrint5-esimerkki NetBeans-projektista Eclipseen seuraavasti:

1. Klikataan Eclipsessä hiiren kakkosnäppäimellä BluePrint5-projektin alla olevaa bpelContent-hakemistoa ja valitaan ”Import”.
2. Valitaan General-kansiosta ”File System” ja painetaan ”Next”.
3. Painetaan Browse-nappulaa, valitaan NetBeansin BluePrint5-projektinalta src-hakemisto ja painetaan ”OK”.
4. Valitaan kaikki src-hakemiston tiedostot ja painetaan ”Finish”.

Eclipsen BluePrint5-projektissa tulisi nyt olla kaikki taulukossa 3 luetellut tiedostot. Kaikkien WSDL-tiedostojen kohdalla näkyy virheilmoitus kertoen validaatiovirheistä (kuva 8).



Kuva 8. BPEL-projektin BluePrint5 validaatiovirheet

5.4.8 Validaatiovirheiden korjaaminen

BPEL Designer -plugin antaa kaikille rajapintakuvauksille saman virheilmoituksen: ”The XPath segment "child::ns:reserve" cannot be resolved.”. Virheilmoitus liittyy WS-BPEL:in propertyAlias-laajennukseen, jonka avulla määritellään korrelaatiossa käytettävän reservationID:n sijainti varausviestissä (koodiesimerkki 5). Virheilmoitus johtuu siitä, että BPEL Designer:in XPath-validointi ei salli XPath-polun sisältävän käsiteltävän viestin juurielementtiä. Kun XPath-polun alusta poistetaan juurielementti, esim. Reservation.wsdl:ssä ”/ns:reserve”, menee validaatio läpi. Korjaus tehdään kaikille rajapintakuvauksille.

```
<bpws:propertyAlias part="reservePart"
                    propertyName="tns:reservationID"
                    messageType="tns:reserveMesg">
        <bpws:query>/ns:reserve/ns:reservationID</bpws:query>
</bpws:propertyAlias>
```

Koodiesimerkki 5. Property-aliaksen määrittäminen, Reservation.wsdl

5.4.9 Rajapintakuvauksien muuttaminen vastaamaan prosessien uutta sijaintia

Koska BluePrint5-esimerkissä ei käytetä dynaamisesti määriteltäviä WS-palveluiden osoitteita (ts. UDDI), täytyy rajapintakuvaukset päivittää vastaamaan palveluiden uutta sijaintia Apache ODE:n alla.

Esimerkiksi ReservationSystem-prosessin osoite oli GlassFish ESB:ssä:

http://localhost:18181/wsdl/ReservationService/ServiceLocation

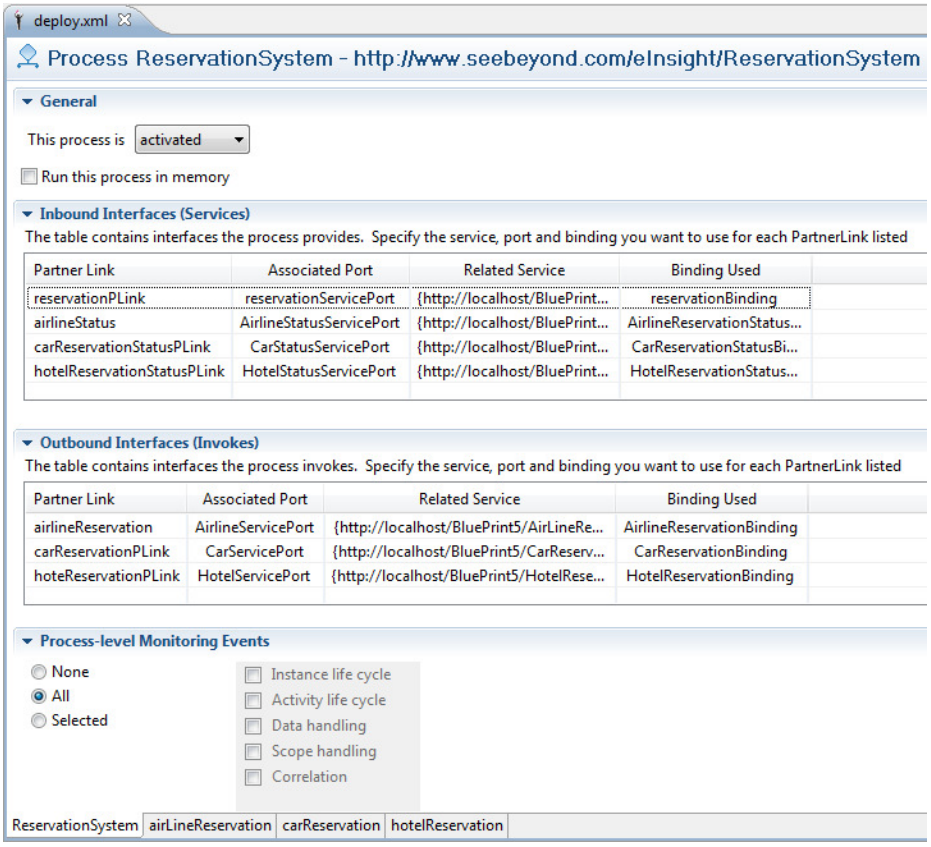
Apache ODE:n alla palvelun osoite on:

http://localhost:8080/processes/reservationService

5.4.10 Esimerkin asentaminen palvelimelle

BPEL-projektin asentaminen Apache ODE:lle vaatii asennuskuvauksen (Deployment Descriptor), joka määritellään erillisessä deploy.xml-tiedostossa. BPEL Designer –pluginissa on oma editorinsa kuvauksen tekemistä varten, mikä helpottaa työskentelyä. Uusi deploy.xml luodaan seuraavasti:

1. Valitaan BluePrint5-projektin kontekstivalikosta New -> Other
2. Valitaan BPEL 2.0 -kansiosta ”Apache ODE Deployment Descriptor”
3. Määritellään editorin avulla projektin kaikkien prosessien partnerLink-kuvauksia vastaavat portType-kuvaukset (kuva 9).
4. Asennetaan BluePrint5-projekti ApacheODE:lle klikkaamalla hiiren kakkosnäppäimellä Server-näkymässä palvelinta ”Ode v1.x Server” ja valitsemalla ”Add and Remove”. Esiin tulevassa asennusvelhossa siirretään BluePrint5-projekti Available-puolelta Configured-puolelle.
5. Käynnistetään palvelin valitsemalla palvelimen kontekstivalikosta ”Start”.



Kuva 9. Apache ODE:n asennuskuvaus, deploy.xml

Palvelin käynnistyy, mutta projektin asennus keskeytyy seuraavaan virheeseen:

```
[PropertyDeclaredWithComplexType] Attempt to declare property "{http://localhost/BluePrint5/AirLineReservation}air" with complex type "{http://ReservationSystem.org/xsd/reserve}reserveType".
```

BluePrint5-esimerkin aliprosessien rajapintakuvauksissa on määritelty WS-BPEL:in property-laajennuksella ominaisuuksia, jotka ovat XML:n complex-tyyppiä. Virheilmoitus antaa ymmärtää, että ne eivät olisi tuettuja. Internetin keskustelupalstoilta löytyi vahvistus: Apache ODE tukee ainoastaan XML simple type -tyyppisiä ominaisuuksia [28; 29].

5.4.11 Kompleksityyppisten property-viittausten poisto

Kun Blueprint5-esimerkin aliprosesseja tutkittiin tarkemmin, havaittiin että ongelmallisia complex-tyyppisiä ominaisuuksia ei käytetä mihinkään. Ainoastaan Reservation.wsdl:n reservationID-property on käytössä korrelaatiossa.

Tämän takia ongelma päätettiin korjata poistamalla ongelmalliset ominaisuudet rajapintakuvauksista (koodiesimerkki 6).

```
<bpws:property name="air" type="ns:reserveType"/>

<bpws:propertyAlias propertyName="tns:air"
                    messageType="tns:AirlineReservation"
                    part="reserveAirlinePart"/>
```

Koodiesimerkki 6. Kompleksityyppinen property ja propertyAlias rajapintakuvauksessa AirLineReservation.wsdl

Seuraavaksi BluePrint5-projekti asennetaan uudestaan ja käynnistetään palvelin uudelleen. Jos halutaan varmistaa, että kaikki tiedostot asennetaan uudelleen, kannattaa ennen uudelleenasennusta poistaa vanha asennus Apache ODE:n alta hakemistosta WEB-INF\processes\BluePrint5.

5.4.12 Testaus

Kun BluePrint5-projekti on saatu asennettua palvelimelle, tulee vielä testata sen toimivuus. Testauksessa voidaan käyttää Eclipsen JavaEE- ja SOA Platform -versioiden mukana tulevaa WebServices Explorer -työkalua.

1. Lisätään WebServices Explorer BPEL-perspektiiviin (päävalikosta Window -> Customize Perspective, välilehti ”Command Groups Availability”).
2. Valitaan päävalikosta Run -> ”Launch the WebServices Explorer”.
3. Valitaan oikeasta yläkulmasta painike “WSDL Page”.
4. Valitaan Navigator-kentässä ”WSDL Main”.
5. Actions-kentässä asetetaan selailtava rajapintakuvaus Browse-linkkiä painamalla.
6. WebBrowser-velho ehdottaa automaattisesti Reservation.wsdl:ää. Painetaan ”Go”.
7. Actions-kentässä on nyt Reservation.wsdl:n URL. Painetaan “Go”.
8. Actions-kenttään listautuu rajapintakuvauksen operaatiot, tässä tapauksessa vain “reserve”. Painetaan sitä.

9. Actions-kenttään ilmestyvät kentät reserve-operaation parametreille, ”reservationID” ja ”Description”.

10. Testataan palvelua käyttämällä taulukon 4 testitapausta 1. Syötetään arvot kenttiin ja painetaan ”Go”.

Status-kenttään tulee WebServices Explorerin ilmoitus:

> “The content of the body cannot be displayed in the form view. Please switch to the source view to examine the raw content.”

Status-kentän source-linkkiä painamalla nähdään paluuviestin lähdekoodi (koodiesimerkki 7). Paluuviesti on Apache ODE:n palauttama virheilmoitus. Se sisältää virhekoodin ”soapenv:Server” ja virheen kuvauksen ”axis2ns2:selectionFailure”.

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
   <soapenv:Fault>
     <faultcode>soapenv:Server</faultcode>
     <faultstring
        xmlns:axis2ns2=
        "http://docs.oasis-open.org/wsbpel/2.0/process/executable">
         axis2ns2:selectionFailure
     </faultstring>
     <detail />
   </soapenv:Fault>
 </soapenv:Body>
</soapenv:Envelope>
```

Koodiesimerkki 7. ReservationServicen palauttama virheilmoitus

Apache ODE:n konsolille ei ilmestynyt mitään virheilmoitusta. Virheilmoitus saadaan näkymään konsolille, kun Apache ODE:n lokitusasetuksiin (WEB-INF\classes\logger.properties) asetetaan seuraava rivi:

```
log4j.category.org.apache.ode=INFO
```

Palvelimen uudelleenkäynnistämisen jälkeen konsolille saadaan seuraava virheilmoitus:

```
[ASSIGN] Assignment Fault:

{http://docs.oasis-
open.org/wsbpel/2.0/process/executable}selectionFailure,

lineNo=104,faultExplanation=No results for expression:

{OXPath10Expression $reserveAirline-
Var.reserveAirlinePart/rsrv:reservationID}
```

Virhe liittyy konsolin virheilmoituksen mukaan assign-operaatioon.

5.4.13 Muuttujien alustaminen assign-operaatiossa

Apache ODE:n nettisivuilta (http://ode.apache.org/assign.html) löytyi tieto siitä, että jos assign-operaation kohteena oleva muuttuja ei palauta lValue-arvoa, toisin sanoen sitä ei ole alustettu, heitetään (throw) kyseinen virheilmoitus. Vahvistus asialle löytyi OASIS:n WS-BPEL-spesifikaation kappaleesta 8.4 Assignment [13]:

> ”A to-spec MUST return an lvalue. If a to-spec does not return an lvalue then a bpel:selectionFailure MUST be thrown.”

Eräältä Internet-keskustelupalstalta [30] puolestaan saatiin selville, että jos assign-operaation kohteena olevaa muuttujaa ei ole alustettu, tekee GlassFish ESB sen automaattisesti. Tässä tapauksessa Apache ODE on kuitenkin enemmän WS-BPEL 2.0 -spesifikaation mukainen ja heittää tilanteessa selectionFailure-virheilmoituksen.

WS-BPEL-muuttujien alustaminen ei ole aivan yksinkertaista, sillä muuttujat ovat yleensä kompleksityyppisiä sanomia. Koodiesimerkissä 8 on esitetty sendCarStatus-muuttujan alustaminen ennen varsinaista kopiointia assign-elementin sisällä.

```
<copy>
  <from>
    <literal>
       <rsrv:reserveStatus
             xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
          <rsrv:reservationID/>
          <rsrv:reservationStatus>false</rsrv:reservationStatus>
       </rsrv:reserveStatus>
    </literal>
  </from>
  <to part="status" variable="sendCarStatus"/>
</copy>
```

Koodiesimerkki 8. Muuttujan sendCarStatus alustus prosessissa carReservation

Samalla tavalla tulee siis alustaa kaikki assign-operaatioiden kohteena olevat muuttujat. Alustamisen seurauksena muuttujan ”sendCarStatus” status-osiossa on jo arvoja, jolloin varsinainen assign-operaatio voidaan tehdä XPath-kyselyä käyttäen (koodiesimerkki 9).

```
<copy>
  <from>$reserveCar.reserveCarPart/rsrv:reservationID</from>
  <to>$sendCarStatus.status/rsrv:reservationID</to>
</copy>
```

Koodiesimerkki 9. Muuttujan käsittelyä XPath-kyselyllä assign-operaatiossa

5.4.14 Testauksen lopputulokset

Kun kaikki assign-operaatioissa käytettävät muuttujat on saatu alustettua, voidaan aloittaa testaus uudestaan luvussa 5.4.12 kuvatulla tavalla. Tällä kertaa virheilmoitusta ei tule, vaan ReservationSystem-prosessi palauttaa samanlaisen vastauksen kuin GlassFish ESB:llä testattaessa. Taulukossa 5 on lueteltu testitulokset samoille testitapauksille, jotka aikaisemmin suoritettiin GlassFish ESB:llä (kts. taulukko 4).

Taulukko 5. BluePrint5-testitapaukset Apache ODE:lla

Testi	Varausviesti	Vastaus
1	reservationID: 0000098762 description: book_airline	airlineStatus: true hotelStatus: false carStatus: false
2	reservationID: 0000098761 description: book_car, book_hotel, book_airline	airlineStatus: true hotelStatus: true carStatus: true
3	reservationID: 0000098762 description: book_hotel	airlineStatus: false hotelStatus: true carStatus: false

Kuten voidaan huomata, ovat testitulokset täsmälleen samat kuin GlassFishESB:llä. Näin voidaan todeta BluePrint5-esimerkin siirtäminen Apache ODE:lle onnistuneeksi.

5.5 Havaintoja siirrettävyydestä

BluePrint5-prosessia siirrettäessä GlassFish ESB:stä Apache ODE:lle kävi ilmi, kuinka tärkeää standardien noudattaminen on. Eri toimittajien valmistamat WS-BPEL-moottorit tarjoavat omia ratkaisujaan ”helpottamaan” elämää, mutta juuri näiden standardista poikkeavien ominaisuuksien käyttäminen aiheuttaa ongelmia kun halutaan siirtyä tuotteesta toiseen.

WS-BPEL-standardi on itsessään kypsä liiketoimintaprosessien siirtämiseksi eri toimittajien tuotteiden välillä, mutta liiketoimintaprosessien kehittäjien on kiinnitettävä paljon huomiota siihen, että pysytään tiukasti kiinni WS-BPEL

2.0 standardissa, eikä käytetä standardin ulkopuolisia ratkaisuja. Tämä on ainoa tapa varmistua siitä, että tuotteesta toiseen siirtyminen sujuu ongelmitta.

6 YHTEENVETO

Tämän työn tarkoituksena oli toimia esiselvityksenä WS-BPEL-liiketoimintaprosessien käytettävyydelle eräässä työn tilaajan asiakasprojektissa. Työssä käytiin läpi WS-BPEL:in suhdetta SOA-arkkitehtuuriin, käytiin läpi SOA:n suunnitteluperiaatteita ja vertailtiin sitä perinteisiin tapoihin toteuttaa hajautettuja järjestelmiä.

SOA:n voidaan sanoa olevan arkkitehtuuri, joka perustuu palvelukeskeisille suunnitteluperiaatteille. SOA:n rakennuskivinä ovat mahdollisimman yleiskäyttöiset palvelut, jotka eivät pidä sisällään yksittäiseen liiketoimintaprosessiin liittyvää liiketoimintalogiikkaa. Liiketoimintaprosessikohtainen logiikka voidaan toteuttaa WS-BPEL-kuvauskieltä käyttämällä orkestrointipalvelussa, joka käyttää yleiskäyttöisiä palveluita toteuttamaan tarvitsemansa tehtävät. Yleiskäyttöisiä palveluita uudelleen käyttämällä saavutetaan tuntuvia kustannussäästöjä päällekkäisyyden poistumisen ja ylläpidettävyyden muodossa.

WS-BPEL on IBM:n ja Microsoftin yhteistyössä kehittämä XML-pohjainen kieli suoritettavien liiketoimintaprosessien kuvaamiseen. Sen uusin versio on OASIS:n vuonna 2007 standardoima WS-BPEL 2.0. Kieltä voi käyttää sekä järjestelmien integroimiseen Web Services -rajapinnan yli että SOA:n orkestrointipalvelukerroksen toteuttavana teknologiana.

Työn tilaajaa kiinnosti erityisesti WS-BPEL-liiketoimintaprosessien siirrettävyys eri toimittajien WS-BPEL-moottoreiden välillä. Osana tätä työtä toteutettiin BluePrint5-esimerkkiprosessin siirtäminen Oraclen GlassFish ESB:ltä Apachen ODE:lle. Todettiin, että siirrettävässä prosessissa oli tukeuduttu GlassFish ESB:n standardista poikkeavaan automaattiseen muuttujien alustamiseen. Toisaalta Apache ODE ei taas tukenut kompleksityyppisiä ominaisuuksia, jotka ovat standardinmukaisia. Näistä eroavaisuuksista johtuen esimerkkiprosessia ei suoraan saatu siirrettyä Apache ODE:lle. Pienehköjä korjauksia tekemällä liiketoimintaprosessi saatiin kuitenkin lopulta siirrettyä onnistuneesti WS-BPEL-moottoreiden välillä.

VIITELUETTELO

[1] Wikipedia, Common Object Request Broker Architecture [verkkodokumentti, viitattu 7.2.2009]. Saatavissa: http://en.wikipedia.org/wiki/CORBA.

[2] Chandran, Praveen - Poduval Arun, Adding BPEL to the Enterprise Integration Mix. 2005. [verkkodokumentti, viitattu 31.1.2009]. Saatavissa: http://www.oracle.com/technology/pub/articles/bpel_cookbook/chandran.html .

[3] Cobban, Michael, What is BPEL and why is it so important to my business? 2004. [verkkodokumentti, viitattu 14.2.2009]. Saatavissa: http://www.softcare.com/whitepapers/wp_whatis_bpel.php.

[4] Hurwitz, Judith ym., *SOA for Dummies*. Wiley Publishing Inc. 2007.

[5] Erl, Thomas, *SOA Principles of Service Design.* Crawfordswille: Prentice Hall. 2008.

[6] Erl, Thomas, *SOA Design Patterns*. Crawfordswille: Prentice Hall. 2009.

[7] OASIS, Reference Model for Service Oriented Architecture 1.0. 12.10.2006. [verkkodokumentti, viitattu 19.2.2011] Saatavissa: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[8] Open Group, The SOA Work Group: Definition of SOA. 2006. [verkkodokumentti, viitattu 19.2.2011] Saatavissa: http://www.opengroup.org/soa/soa/def.htm.

[9] Erl, Thomas, *Service-Oriented Architecture: Concepts, Technology, and Design*. Crawfordswille: Prentice Hall. 2005.

[10] Erl, Thomas: SOA-analysis. [verkkodokumentti, viitattu 12.3.2011] Saatavissa: http://www.whatissoa.com/p30.php.

[11] Park, Sihyung, The Building Blocks of SOA. 5.2.2010. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://soamag.com/I36/0210-1.php.

[12] Eric Roch, The Service Oriented Architecture (SOA) Blog: SOA Data Services. 3.4.2008. [verkkodokumentti, viitattu 19.2.2011] Saatavissa: http://it.toolbox.com/blogs/the-soa-blog/soa-data-services-23515.

[13] Bharti, Nitin, Web Service Contract Versioning Fundamentals Part I: Versioning and Compatibility. 24.11.2008. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://soa.dzone.com/articles/web-service-contract-versionin.

[14] Juric, Matjaz B., A Hands-on Introduction to BPEL. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html.

[15] Leppänen, Tony, Business Process Execution Language (BPEL). 2006. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: www.automationit.hut.fi/file.php?id=642.

[16] OASIS, Web Service Business Process Execution Language Version 2.0. 11.4.2007. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[17] Smith, Howard, Enough is enough in the field of BPM: We don't need BPELJ: BPML semantics are just fine. 2004. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://www.fairdene.com/bpelj/BPELJ-Enough-Is-Enough.pdf.

[18] Wikipedia, Business Process Modeling Language. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://en.wikipedia.org/wiki/Business_Process_Modeling_Language.

[19] Wainewright, Phil, Loosely Coupled, Small-minded orchestration. 26.3.2004. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://www.looselycoupled.com/blog/lc00aa00025.html.

[20] Dubray, Jean-Jacques, Update. 22.1.2008. [verkkodokumentti, viitattu 16.5.2010.] Saatavissa: http://www.ebpml.org/bpml.html.

[21] Eric Newcomer, Greg Lomow. 14.12.2004. Understanding SOA with Web Services. (eBook). Orchestration and Choreography Specifications. http://book.chinaunix.net/special/ebook/addison/UnderstandingSOAwithWebServices/0321180860/ch06lev1sec4.html.

[22] Khalaf, Rania - Roller, Dieter - Leymann Frank, Revisiting the Behavior of Fault and Compensation Handlers in WS-BPEL. 2009. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://domino.research.ibm.com/comm/research_people.nsf/pages/rkhalaf.pubs.html/$FILE/khalaf_coopis09Paper.pdf.

[23] Wikipedia, Business Process Modeling Notation. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://en.wikipedia.org/wiki/Business_Process_Modeling_Notation.

[24] Dubray, Jean-Jacques, The Seven Fallacies of Business Process Execution. 4.12.2007. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://www.infoq.com/articles/seven-fallacies-of-bpm.

[25] Netbeans, Service Orchestration Functional Specification - Overview. 29.10.2009. [verkkodokumentti, viitattu 16.5.2010] Saatavissa: http://soa.netbeans.org/specs/bpel/orchestrationOverview.html.

[26] Seeley, Rich, Online real estate SOA moves into BPEL. 3.4.2008. [verkkodokumentti, viitattu 31.1.2009]. Saatavissa: http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1308224,00.html.

[27] Seeley, Rich, Verizon uses BPEL app to cut down on code, check for fraud, and go green. 24.9.2008. [verkkodokumentti, viitattu 31.1.2009] Saatavissa: http://searchsoa.techtarget.com/news/1331852/Verizon-uses-BPEL-app-to-cut-down-on-code-check-for-fraud-and-go-green.

[28] Seeley, Rich, SOA, Web Services and BPEL converge at AT&T subsidiary. 20.11.2006. [verkkodokumentti, viitattu 31.1.2009]. Saatavissa:

http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1230792,00.html.

[29] Chandran Anup, Correlation : Unable to evaluate apply property alias , 26.9.2007. [verkkodokumentti, viitattu 19.3.2011] Saatavissa: http://www.mail-archive.com/user@ode.apache.org/msg00309.html.

[30] Ye, Chunyang, Ode correlation does not work with more than one element in the input message. 1.7.2010. [verkkodokumentti, viitattu 19.3.2011] Saatavissa: http://blog.gmane.org/gmane.comp.apache.ode.user/month=20100701.

[31] Nabble.com, OpenESB Users, Question about initialize variables. 10.2008. [verkkodokumentti, viitattu 20.3.2011] Saatavissa: http://openesb-users.794670.n2.nabble.com/Questionabout-initialize-variables-td1332389.html.

LIITE 1: WS-BPEL-PROSESSIN RAJAPINTAKUVAUS (RESERVATION.WSDL)

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
  Copyright (c) 2007, Sun Microsystems, Inc. All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions are met:

  * Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the documentation
    and/or other materials provided with the distribution.

  * Neither the name of Sun Microsystems, Inc. nor the names of its contributors
    may be used to endorse or promote products derived from this software without
    specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
  THE POSSIBILITY OF SUCH DAMAGE.
-->
<definitions targetNamespace="http://localhost/BluePrint5/Reservation"
             name="Reservation"
             xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/varprop"
             xmlns:tns="http://localhost/BluePrint5/Reservation"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:plink="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
   xmlns:ns="http://ReservationSystem.org/xsd/reserve">
    <types>
        <xsd:schema targetNamespace="http://localhost/BluePrint5/Reservation">
            <xsd:import schemaLocation="reservation.xsd"
namespace="http://ReservationSystem.org/xsd/reserve"/>
            <xsd:element name="faultString" type="xsd:string"/>
        </xsd:schema>
    </types>

    <message name="reserveMesg">
        <part name="reservePart" element="ns:reserve"></part>
    </message>

    <message name="reservationStatus">
        <part name="reservationStatusPart" element="ns:reservationStatusOfAll"></part>
    </message>

    <message name="reservationFailed">
        <part name="faultInfo" element="tns:faultString"></part>
    </message>

    <portType name="doReservePortType">
        <operation name="reserve">
            <input name="reserveMesg" message="tns:reserveMesg"></input>
            <output name="reservationStatus" message="tns:reservationStatus"></output>
            <fault name="reservationFailed" message="tns:reservationFailed"></fault>
        </operation>
    </portType>

    <binding name="reservationBinding" type="tns:doReservePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="reserve">
            <input name="reserveMesg">
                <soap:body parts="reservePart" use="literal"/>
            </input>
```

```
            <output name="reservationStatus">
                <soap:body parts="reservationStatusPart" use="literal"/>
            </output>
            <fault name="reservationFailed">
                <soap:fault name="reservationFailed" use="literal"/>
            </fault>
        </operation>
    </binding>

    <service name="reservationService">
        <port name="reservationServicePort"
              binding="tns:reservationBinding">
            <soap:address
location="http://localhost:8080/ode/processes/ReservationService"/>
        </port>
    </service>

    <plink:partnerLinkType name="reservationPLinkType">
        <plink:role name="reservationServicing"
portType="tns:doReservePortType"></plink:role>
    </plink:partnerLinkType>

    <bpws:property name="reservationID" type="xsd:long"></bpws:property>

    <bpws:propertyAlias part="reservePart"
                        propertyName="tns:reservationID"
                        messageType="tns:reserveMesg">
        <bpws:query>/ns:reservationID</bpws:query>
    </bpws:propertyAlias>

</definitions>
```

LIITE 2: WS-BPEL-PROSESSI (RESERVATIONSYSTEM.BPEL)

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Copyright (c) 2007, Sun Microsystems, Inc. All rights reserved. Redistribution
 and use in source and binary forms, with or without modification, are permitted
 provided that the following conditions are met: * Redistributions of source
 code must retain the above copyright notice, this list of conditions and
 the following disclaimer. * Redistributions in binary form must reproduce
 the above copyright notice, this list of conditions and the following disclaimer
 in the documentation and/or other materials provided with the distribution.
 * Neither the name of Sun Microsystems, Inc. nor the names of its contributors
 may be used to endorse or promote products derived from this software without
 specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
 HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
 BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 OF SUCH DAMAGE. -->
<process name="ReservationSystem"
 targetNamespace="http://www.seebeyond.com/eInsight/ReservationSystem"
 xmlns:tns="http://www.seebeyond.com/eInsight/ReservationSystem"
 xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
 xmlns:ns0="http://localhost/BluePrint5/Reservation"
xmlns:ns1="http://localhost/BluePrint5/AirLineReservation"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
 xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
 xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"
 xmlns:sxeh="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/ErrorHandling"
 xmlns:ns3="http://localhost/BluePrint5/HotelReservation"
xmlns:ns2="http://localhost/BluePrint5/CarReservation"
 xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">

 <import namespace="http://localhost/BluePrint5/Reservation"
  location="Reservation.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></import>

 <import namespace="http://localhost/BluePrint5/AirLineReservation"
  location="AirLineReservation.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"></import>

 <import namespace="http://localhost/BluePrint5/CarReservation"
  location="CarReservation.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></import>

 <import namespace="http://localhost/BluePrint5/HotelReservation"
  location="HotelReservation.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></import>

 <partnerLinks>
  <partnerLink name="reservationPLink" partnerLinkType="ns0:reservationPLinkType"
   myRole="reservationServicing" />
  <partnerLink name="airlineReservation" partnerLinkType="ns1:AirlineReservationPLinkType"
   partnerRole="reservingRole" />
  <partnerLink name="airlineStatus" partnerLinkType="ns1:AirLineReservStatusPLinkType"
   myRole="statusReceivingRole" />
  <partnerLink name="carReservationPLink" partnerLinkType="ns2:CarReservationPLinkType"
   partnerRole="reservingRole" />
  <partnerLink name="carReservationStatusPLink"
   partnerLinkType="ns2:CarReservStatusPLinkType" myRole="statusReceivingRole" />
  <partnerLink name="hoteReservationPLink" partnerLinkType="ns3:HotelReservationPLinkType"
   partnerRole="reservingRole" />
  <partnerLink name="hotelReservationStatusPLink"
   partnerLinkType="ns3:HotelReservStatusPLinkType" myRole="statusReceivingRole" />
 </partnerLinks>

 <variables>
  <variable name="reserve" messageType="ns0:reserveMesg"></variable>
  <variable name="reservationStatus" messageType="ns0:reservationStatus"></variable>
  <variable name="reserveAirlineVar" messageType="ns1:AirlineReservation"></variable>
  <variable name="airlineStatusVar" messageType="ns1:AirlineReservationStatus"></variable>
  <variable name="reserveCarVar" messageType="ns2:CarReservation"></variable>
  <variable name="carStatusVar" messageType="ns2:CarReservationStatus"></variable>
```

```
  <variable name="reserveHotelVar" messageType="ns3:HotelReservation"></variable>
  <variable name="hotelStatusVar" messageType="ns3:HotelReservationStatus"></variable>
  <variable name="tempCarStatusVar" messageType="ns2:CarReservationStatus"></variable>
  <variable name="tempAirlineStatusVar"
messageType="ns1:AirlineReservationStatus"></variable>
  <variable name="tempHotelStatusVar" messageType="ns3:HotelReservationStatus"></variable>
 </variables>

 <correlationSets>
  <correlationSet name="reservationCorrelationSet"
   properties="ns0:reservationID " />
 </correlationSets>

 <sequence>
  <receive name="reserve" partnerLink="reservationPLink"
   portType="ns0:doReservePortType" operation="reserve" variable="reserve"
   createInstance="yes">
   <correlations>
    <correlation set="reservationCorrelationSet" initiate="yes" />
   </correlations>
  </receive>

  <assign name="Assign_1">
   <copy>
    <from>
     <literal>
      <rsrv:reserve xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
       <rsrv:reservationID />
       <rsrv:description />
      </rsrv:reserve>
     </literal>
    </from>
    <to part="reserveAirlinePart" variable="reserveAirlineVar" />
   </copy>
   <copy>
    <from>
     <literal>
      <rsrv:reserve xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
       <rsrv:reservationID />
       <rsrv:description />
      </rsrv:reserve>
     </literal>
    </from>
    <to part="reserveCarPart" variable="reserveCarVar" />
   </copy>
   <copy>
    <from>
     <literal>
      <rsrv:reserve xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
       <rsrv:reservationID />
       <rsrv:description />
      </rsrv:reserve>
     </literal>
    </from>
    <to part="reserveHotelPart" variable="reserveHotelVar" />
   </copy>

   <copy>
    <from>$reserve.reservePart/rsrv:description</from>
    <to>$reserveAirlineVar.reserveAirlinePart/rsrv:description</to>
   </copy>
   <copy>
    <from>$reserve.reservePart/rsrv:reservationID</from>
    <to>$reserveCarVar.reserveCarPart/rsrv:reservationID</to>
   </copy>
   <copy>
    <from>$reserve.reservePart/rsrv:reservationID</from>
    <to>$reserveHotelVar.reserveHotelPart/rsrv:reservationID</to>
   </copy>

   <copy>
    <from>$reserve.reservePart/rsrv:reservationID</from>
    <to>$reserveAirlineVar.reserveAirlinePart/rsrv:reservationID</to>
   </copy>
   <copy>
    <from>$reserve.reservePart/rsrv:description</from>
    <to>$reserveAirlineVar.reserveAirlinePart/rsrv:description</to>
```

```
 </copy>
 <copy>
  <from>$reserve.reservePart/rsrv:reservationID</from>
  <to>$reserveCarVar.reserveCarPart/rsrv:reservationID</to>
 </copy>
 <copy>
  <from>$reserve.reservePart/rsrv:description</from>
  <to>$reserveCarVar.reserveCarPart/rsrv:description</to>
 </copy>
 <copy>
  <from>$reserve.reservePart/rsrv:reservationID</from>
  <to>$reserveHotelVar.reserveHotelPart/rsrv:reservationID</to>
 </copy>
 <copy>
  <from>$reserve.reservePart/rsrv:description</from>
  <to>$reserveHotelVar.reserveHotelPart/rsrv:description</to>
 </copy>
</assign>

<flow name="Flow2">
 <invoke name="reserveAirline" partnerLink="airlineReservation"
  portType="ns1:AirlineReservationPortType" operation="reserveAirline"
  inputVariable="reserveAirlineVar"></invoke>
 <invoke name="reserveCar" partnerLink="carReservationPLink"
  portType="ns2:CarReservationPortType" operation="reserveCar"
  inputVariable="reserveCarVar"></invoke>
 <invoke name="reserveHotel" partnerLink="hoteReservationPLink"
  portType="ns3:HotelReservationPortType" operation="reserveHotel"
  inputVariable="reserveHotelVar"></invoke>
</flow>
<flow name="Flow">
 <pick name="receiveHotelStatus">
  <onMessage partnerLink="hotelReservationStatusPLink"
   portType="ns3:HotelStatusPortType" operation="sendHotelStatus"
   variable="hotelStatusVar">
   <correlations>
    <correlation set="reservationCorrelationSet" initiate="no" />
   </correlations>
   <sequence>
    <empty name="Empty3"></empty>
    <assign name="Assign_7">
     <copy>
      <from>
       <literal>
        <rsrv:reserveStatus xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
         <rsrv:reservationID />
         <rsrv:reservationStatus>0</rsrv:reservationStatus>
        </rsrv:reserveStatus>
       </literal>
      </from>
      <to>$tempHotelStatusVar.status</to>
     </copy>
     <copy>
      <from>$hotelStatusVar.status/rsrv:reservationStatus</from>
      <to>$tempHotelStatusVar.status/rsrv:reservationStatus</to>
     </copy>
    </assign>
   </sequence>
  </onMessage>
  <onAlarm>
   <for>'P0Y0DT2M'</for>
   <assign name="Assign_2">
    <copy>
     <from>false()</from>
     <to>$tempHotelStatusVar.status/rsrv:reservationStatus</to>
    </copy>
   </assign>
  </onAlarm>
 </pick>
 <pick name="receiveCarStatus">
  <onMessage partnerLink="carReservationStatusPLink"
   portType="ns2:CarStatusPortType" operation="sendCarStatus"
   variable="carStatusVar">
   <correlations>
    <correlation set="reservationCorrelationSet" initiate="no" />
   </correlations>
   <sequence>
```

```
      <empty name="Empty_2"></empty>
      <assign name="Assign1">
       <copy>
        <from>
         <literal>
          <rsrv:reserveStatus xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
           <rsrv:reservationID />
           <rsrv:reservationStatus>0</rsrv:reservationStatus>
          </rsrv:reserveStatus>
         </literal>
        </from>
        <to>$tempCarStatusVar.status</to>
       </copy>
       <copy>
        <from>$carStatusVar.status/rsrv:reservationStatus</from>
        <to>$tempCarStatusVar.status/rsrv:reservationStatus</to>
       </copy>
      </assign>
     </sequence>
    </onMessage>
    <onAlarm>
     <for>'P0Y0DT2M'</for>
     <assign name="Assign_3">
      <copy>
       <from>false()</from>
       <to>$tempCarStatusVar.status/rsrv:reservationStatus</to>
      </copy>
     </assign>
    </onAlarm>
   </pick>
   <pick name="receiveAirlineStatus">
    <onMessage partnerLink="airlineStatus" portType="ns1:AirlineStatusPortType"
     operation="sendAirlineStatus" variable="airlineStatusVar">
     <correlations>
      <correlation set="reservationCorrelationSet" initiate="no" />
     </correlations>
     <sequence>
      <empty name="Empty"></empty>
      <assign name="Assign">
       <copy>
        <from>
         <literal>
          <rsrv:reserveStatus xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
           <rsrv:reservationID />
           <rsrv:reservationStatus>0</rsrv:reservationStatus>
          </rsrv:reserveStatus>
         </literal>
        </from>
        <to>$tempAirlineStatusVar.status</to>
       </copy>
       <copy>
        <from>$airlineStatusVar.status/rsrv:reservationStatus</from>
        <to>$tempAirlineStatusVar.status/rsrv:reservationStatus</to>
       </copy>
      </assign>
     </sequence>
    </onMessage>
    <onAlarm>
     <for>'P0Y0DT2M'</for>
     <assign name="Assign_4">
      <copy>
       <from>false()</from>
       <to>$tempAirlineStatusVar.status/rsrv:reservationStatus</to>
      </copy>
     </assign>
    </onAlarm>
   </pick>
  </flow>

  <assign name="Assign2">
   <copy>
    <from>
     <literal>
      <rsrv:reservationStatusOfAll xmlns:rsrv="http://ReservationSystem.org/xsd/reserve">
       <rsrv:airlineStatus>0</rsrv:airlineStatus>
       <rsrv:hotelStatus>0</rsrv:hotelStatus>
       <rsrv:carStatus>0</rsrv:carStatus>
```

```
      </rsrv:reservationStatusOfAll>
     </literal>
    </from>
    <to>$reservationStatus.reservationStatusPart</to>
   </copy>
   <copy>
    <from>$tempAirlineStatusVar.status/rsrv:reservationStatus</from>
    <to>$reservationStatus.reservationStatusPart/rsrv:airlineStatus</to>
   </copy>
   <copy>
    <from>$tempCarStatusVar.status/rsrv:reservationStatus</from>
    <to>$reservationStatus.reservationStatusPart/rsrv:carStatus</to>
   </copy>
   <copy>
    <from>$tempHotelStatusVar.status/rsrv:reservationStatus</from>
    <to>$reservationStatus.reservationStatusPart/rsrv:hotelStatus</to>
   </copy>
  </assign>

  <reply name="reserve.Reply" partnerLink="reservationPLink"
   portType="ns0:doReservePortType" operation="reserve" varia-
ble="reservationStatus"></reply>
 </sequence>
</process>
```

LIITE 3: WS-BPEL 2.0 YLÄTASON ELEMENTIT [13]

Elementti	Kuvaus
Process	WS-BPEL-liiketoimintaprosessi alkaa process-juurielementillä. Se sisältää aina attribuutit prosessin nimelle ja prosessissa esiintyville nimiavaruuksille. Lisäksi voidaan määrittää attribuutit queryLanguage ja expressionLanguage, joilla asetetaan käytettävät kysely- ja lausekekielet. Oletusarvo molemmille on XPath 1.0.
Extensions	WS-BPEL on suunniteltu laajennettavaksi kieleksi. Laajennus voi olla uusi attribuutti, elementti, tai muutos ajonaikaiseen toiminnallisuuteen. Lapsielementillä extension määritellään laajennuksen nimiavaruus ja kerrotaan WS-BPEL-moottorille, onko sen pakko tukea laajennusta vai ei. Jos yksi tai useampi laajennus on asetettu pakolliseksi, mutta moottori ei sitä ymmärrä, prosessi keskeytetään.
Import	Import-elementillä ilmoitetaan WS-BPEL-prosessin riippuvuus ulkopuolisesta XML-skeemasta tai WSDL-määrityksestä. Import-elementillä on yksi pakollinen importType-attribuutti, ja kaksi vapaavalintaista attribuuttia, namespace ja location. Attribuutin importType arvon tulee olla http://www.w3.org/2001/XMLSchema kun linkitetään XML 1.0 -muotoista dokumenttia, ja http://schemas.xmlsoap.org/wsdl/ kun kyseessä on WSDL 1.1 -dokumentti. Jos namespace ja location -attribuutteja ei käytetä, on kysymyksessä nimiavaruudeton dokumentti, minkä sijainnissa luotetaan prosessin ulkopuolisiin määrityksiin.
partnerLinks	Elementin partnerLinks lapsielementtiä partnerLink käytetään mallintamaan WS-BPEL-prosessin kanssa kommunikoivaa palvelua. Jokainen partnerLink-elementti viittaa yhteen partnerLinkType-elementtiin (kts. kappale 3.4.1). Samaan partnerLinkType-elementtiin voidaan viitata useammasta partnerLink-elementistä. Liiketoimintaprosessin rooli määritellään myRole-attribuutilla ja osakasprosessin partnerRole-attribuutilla. Jos elementissä partnerLinkType on määritelty vain yksi rooli, niin tehdään myös partnerLink-elementissä.

messageExchanges	Tätä vapaavalintaista elementtiä käytetään täsmentämään sisään tulevien viestiaktiviteettien (Inbound Message Activity, IMA) ja reply-aktiviteettien välistä suhdetta. IMA-aktiviteetteja ovat receive, pick ja onEvent. MessageExchanges-aktiviteetti on tarpeellinen ainoastaan, jos on samaa partnerLink:iä ja operaatiota kohden voi muodostua ajonaikaisesti useita vastaanotto-vastaus -pareja. Reply-elementti assosioituu IMA-aktiviteettien siis kanssa partnerLinkin, operaation, ja messageExchangen välityksellä.
Variables	Variables-elementissä määriteltyjen muuttujien avulla voidaan tallentaa viestejä, jotka määrittävät omalta osaltaan liiketoiminta-prosessin tilan. Säilytettävät viestit ovat yleisesti joko osakkailta saapuneita tai niille lähteviä. Muuttujissa voi myös säilyttää tilatietoa, jota ei koskaan vaihdeta osakkaiden kanssa. WS-BPEL tarjoaa kolmenlaisia muuttujatyyppejä: WSDL-viesti, XML-skeema, tai XML-skeemaelementti.
correlationSets	WS-BPEL -prosessista voidaan käynnistää useampia instansseja, jotka kommunikoivat osakaspalveluiden kanssa omien elinkaariensa mukaisesti. Jos WS-BPEL-prosessille toimitettavat viestit osoitetaan sille pelkän WSDL-portin perusteella, jää avoimeksi mille prosessi-instanssille viestit tulee ohjata. Poikkeuksena ovat viestit, jotka käynnistävät uuden prosessi-instanssin. WS-BPEL-prosessissa voidaan määritellä correlationSets-elementissä korrelaatiotunnisteita, joiden arvot ovat instanssikohtaisia ja jotka sisällytetään viestien otsikkokenttään.
faultHandlers	Virheenkäsittelijät määritellään faultHandlers-elementin avulla. Virheenkäsittelijöiden tehtävänä on peruuttaa epäonnistuneesti tehty työ näkyvyysalueella (kts. kappale 3.4.6), jossa virhe on tapahtunut. Kompensaatio (kts. kappale 3.4.7) ei ole mahdollista näkyvyysalueella, missä virheenkäsittelijä on lauennut. Virhetilanteet hallitaan catch- ja catchAll-lapsielelementeissä.
eventHandlers	Tapahtumankäsittelijät määritellään eventHandlers-elementin avulla. Useampi tapahtumankäsittelijä voidaan suorittaa samanaikaisesti. Tapahtumia on kahdentyyppisiä: sisään tulevia sanomia ja hälytyksiä, jotka laukeavat ajastettuina.

LIITE 4: WS-BPEL 2.0 PERUSAKTIVITEETIT [13]

Elementti	Kuvaus
Invoke	Invoke-aktiviteetilla kutsutaan osakaspalveluiden operaatioita. Invoke-aktiviteetti voi sisältää myös muita aktiviteetteja virheenkäsittelijän tai kompensaationkäsittelijän sisään käärittynä. Operaatiot voivat olla yksi- tai kaksisuuntaisia WSDL-kuvausten mukaisesti.
Receive	Receive kuuluu IMA-aktiviteetteihin (Inbound Message Activities), jotka tarjoavat palveluita liiketoimintaprosessin osakkaille. Muita IMA-aktiviteetteja ovat pick, onEvent ja onMessage. Receive-aktiviteetti määrittelee käytettävän myRole-roolin sisältävän partnerLink:in ja suoritettavan WSDL-operaation. Porttityypin määrittely on vapaavalintaista. Attribuutilla createInstance ilmaistaan, käynnistetäänkö viestin saapuessa uusi instanssi liiketoimintaprosessista. Käynnistysaktiviteetti on siis receive- tai pick-aktiviteetti, jossa createInstance-attribuutti saa arvon ”yes”. Attribuutin oletusarvo on ”no”.
Reply	Reply-aktiviteettia käytetään lähettämään vastaus IMA-aktiviteetin kautta tulleeseen pyyntösanomaan. Vastausviesti määritellään variable-attribuutilla. Jos reply-aktiviteettia käytetään virheviestin lähettämiseen, käytetään lisäksi faultName-attribuuttia, ja tällöin variable-attribuutin avulla välitetään lähetettävä virhesanoma.
Assign	Assign-aktiviteettia käytetään kopioimaan tietoa muuttujasta toiseen, kuten myös muodostamaan uutta tietoa lausekkeiden avulla. Tiedonmuokkausoperaatioita on myös mahdollista ottaa käyttöön muista kuin WS-BPEL:in omasta nimiavaruudesta.
Throw	Throw-aktiviteettia käytetään tiedottamaan liiketoimintaprosessin sisäisestä virheestä. Aktiviteetti sisältää virheen nimen ja haluttaessa myös tarkempaa tietoa virheestä. Virheenkäsittelijä voi käyttää tätä tietoa hyväkseen käsitellessään virhettä ja populoidessaan virheviestejä muille palveluille.
Wait	Wait-aktiviteetin avulla voidaan odottaa tietyn ajanjakson verran tai tiettyyn ajanhetkeen asti.

Empty	Monesti on tarpeen käyttää aktiviteettia, mikä ei tee mitään, esimerkiksi kun tulee ottaa kiinni poikkeus, minkä ei haluta aiheuttavan mitään toimenpidettä. Tätä varten on empty-aktiviteetti.
extensionActivity	extensionActivity-aktviteetin sisään kääritään WS-BPEL:in ulkopuolinen laajennusaktiviteetti. Sen pitää olla yksittäinen ja nimiavaruudeltaan WS-BPEL:istä eroava elementti.
Exit	Exit-aktiviteettia käytetään liiketoimintaprosessin kaikkien suorituksessa olevien aktiviteettien välittömään keskeyttämiseen. Mitään virheenkäsittelyä, lopetuksenkäsittelyä tai kompensaatiokäsittelyä ei tehdä.
Rethrow	Rethrow-aktiviteettia käytetään vain virheenkäsittelijöissä heittämään edelleen kiinniotettu virhe.

LIITE 5: WS-BPEL 2.0 RAKENTEISET AKTIVITEETIT [13]

Elementti	Kuvaus
Sequence	Sequence-aktiviteetin sisään käärityt aktiviteetit suoritetaan peräkkäin siinä järjestyksessä kuin ne esiintyvät sequence-aktiviteetissa.
If	If-aktiviteetti mahdollistaa ehdollisen käyttäytymisen. Ehto ilmoitetaan condition-elementin sisällä joko oletuslausekekielellä, tai jollain laajennuskielellä. Jokainen vaihtoehtoinen elself-elementti aloittaa uuden haaran omine ehtolauseineen. Jos minkään haaran ehto ei täyty, voidaan suorittaa vaihtoehtoisen else-elementin sisältö.
While	While-aktiviteetin avulla voidaan toistuvasti suorittaa aktiviteetin sisältö, niin kauan kuin condition-elementissä kerrottu ehto on tosi. Ehto tarkistetaan jokaisen iteraation alussa.
repeatUntil	RepeatUntil-aktiviteetin avulla voidaan toistuvasti suorittaa aktiviteetin sisältö, niin kauan kuin condition-elementissä kerrottu ehto on epätosi. Ehto tarkistetaan jokaisen iteraation lopussa, joten while-aktiviteetista poiketen luuppi suoritetaan vähintäänkin kerran.
Pick	Pick-aktiviteetti odottaa yhtä tapahtumaa tapahtuvaksi sille ilmoitetusta joukosta tapahtumia. Pick-aktiviteetti koostuu joukosta tapahtuma-aktiviteetti pareja, eli tiettyä tapahtumaa vastaa aina yksi suoritettava aktiviteetti. Kun jokin tapahtuma on tapahtunut, ei muita vaihtoehtoisia tapahtumia enää vastaanoteta pick-aktiviteetin toimesta. Jos useampi tapahtuma saapuu samaan aikaan, syntyy race-poikkeus, jonka käsittely on prosessimoottorin toteutuksesta riippuvainen.
Flow	Flow-aktiviteetti mahdollistaa aktiviteettien samanaikaisen suorituksen ja synkronoinnin. Flow-aktiviteetin suoritus päättyy, kun kaikkien sen sisältämien aktiviteettien suoritus on päättynyt. Myös flow:n aikana suoritettaville aktiviteeteille on mahdollista asettaa keskinäisiä riippuvuuksia link-elementtien avulla, jos halutaan varmistaa että tietty aktiviteetti on suoritettu ennen toisen aloittamista.

forEach	ForEach-aktiviteetti suorittaa sisältämänsä scope-aktiviteetin (kts. luku 3.4.6) täsmälleen N+1 kertaa, missä N on yhtä kuin forEach-aktiviteetin sisäisen finalCounterValue-elementin arvo vähennettynä vastaavan startCounterValue-elementin arvolla.